Predicting Refrigerant Inventory of HFC 134a in Air Cooled Condensers

L. A. Orth and C. O. Pedersen

ACRC TR-50 September 1993

For additional information:

Air Conditioning and Refrigeration Center
University of Illinois
Mechanical & Industrial Engineering Dept.
1206 West Green Street
Urbana, IL 61801

(217) 333-3115

Prepared as part of ACRC Project 03
Condenser Performance
C. O. Pedersen, Principal Investigator

The Air Conditioning and Refrigeration Center was founded in 1988 with a grant from the estate of Richard W. Kritzer, the founder of Peerless of America Inc. A State of Illinois Technology Challenge Grant helped build the laboratory facilities. The ACRC receives continuing support from the Richard W. Kritzer Endowment and the National Science Foundation. The following organizations have also become sponsors of the Center.

Acustar Division of Chrysler
Allied-Signal, Inc.
Amana Refrigeration, Inc.
Brazeway, Inc.
Carrier Corporation
Caterpillar, Inc.
E. I. du Pont de Nemours & Co.
Electric Power Research Institute
Ford Motor Company
Frigidaire Company
General Electric Company
Harrison Division of GM
ICI Americas, Inc.
Modine Manufacturing Co.
Peerless of America, Inc.
Environmental Protection Agency
U. S. Army CERL
Whirlpool Corporation

For additional information:

Air Conditioning & Refrigeration Center
Mechanical & Industrial Engineering Dept.
University of Illinois
1206 West Green Street
Urbana IL 61801

217 333 3115

TABLE OF CONTENTS

NOMENCLATURE

α	Void fraction	
A_i	Inside tube cross-sectional area	ft^2
A_{mean}	Mean tube cross-sectional area	ft^2
$A_{si(i)}$	Module i inside surface area	ft^2
$A_{so(i)}$	Module i outside surface area	ft^2
$C_{air(i)}$	Module i air heat capacity rate	Btu/hr-°F
$C_{max(i)}$	Module i maximum heat capacity rate	Btu/hr-°F
$C_{min(i)}$	Module i minimum heat capacity rate	Btu/hr-°F
$c_{pA(i)}$	Module i air specific heat	Btu/lbm-°F
$c_{p\,air}$	Condenser air specific heat	Btu/lbm-°F
$c_{pf(i)}$	Module i refrigerant liquid specific heat	Btu/lbm-°F
$c_{pR(i)}$	Module i refrigerant specific heat	Btu/lbm-°F
$C_{ref(i)}$	Module i refrigerant heat capacity rate	Btu/hr-°F
$C_{ratio(i)}$	Module i heat capacity ratio	
D_i	Inside tube diameter	ft
D_h	Air side hydraulic diameter	ft
ΔP_{com}	Component refrigerant pressure drop	psi
$\Delta P_{fric(i)}$	Module i frictional refrigerant pressure drop	psi
$\Delta P_{grav(i)}$	Module i gravitational refrigerant pressure drop	psi
$\Delta P_{mom(i)}$	Module i momentum refrigerant pressure drop	psi
$\Delta P_{mod(i)}$	Module i total refrigerant pressure drop	psi
$\varepsilon_{(i)}$	Module i effectiveness	
f	Fanning friction factor	
Fr	Refrigerant Froude number	
$f_Q(x)$	Heat flux assumption	
G_{air}	Air side total mass flux	lbm/ft^2-hr
G_{ref}	Refrigerant mass flux	lbm/ft^2-hr
H	Condenser height	ft
h_{air}	Air side total heat transfer coefficient	Btu/hr-ft^2-°F
$h_{air(i)}$	Module i air side heat transfer coefficient	Btu/hr-ft^2-°F
h_{fg}	Latent heat of vaporization	Btu/lbm
$h_{ref(i)}$	Module i refrigerant side heat transfer coefficient	Btu/hr-ft^2-°F
$h_{Rin(i)}$	Module i refrigerant inlet enthalpy	Btu/lbm

$h_{Rout(i)}$	Module i refrigerant outlet enthalpy	Btu/lbm
j	Colburn j-factor	
k_{air}	Air thermal conductivity	Btu/hr-ft-°F
$k_{f(i)}$	Module i refrigerant liquid thermal conductivity	Btu/hr-ft-°F
$k_{ref(i)}$	Module i refrigerant thermal conductivity	Btu/hr-ft-°F
k_{tube}	Tube thermal conductivity	Btu/hr-ft-°F
L_{cond}	Total refrigerant tube length	ft
$L_{mod(i)}$	Module i length	ft
L_{seg}	Segment length	ft
$m_{liq(i)}$	Module i refrigerant liquid mass	lbm
$m_{mod(i)}$	Module i refrigerant mass	lbm
m_{total}	Total condenser mass	lbm
$m_{vap(i)}$	Module i refrigerant vapor mass	lbm
$\dot{m}_{A\ mod(i)}$	Module i air mass flow rate	lbm/hr
$\dot{m}_{Atot}$	Total air mass flow rate	lbm/hr
$\dot{m}_{R\ mod}$	Module refrigerant mass flow rate	lbm/hr
$\dot{m}_{Rseg}$	Segment refrigerant mass flow rate	lbm/hr
$\dot{m}_{Rtot}$	Total refrigerant mass flow rate	lbm/hr
n	Number of modules in segment	
η_s	Air side overall efficiency	
n_{tubes}	Number of tubes in manifold section	
NTU	Number of transfer units	
Nu_{air}	Air side Nusselt number	
$Nu_{ref(i)}$	Module i refrigerant Nusselt number	
Φ_{vap}	Two phase frictional multiplier	
p	Number of segments in condenser	
Pr_{air}	Air side Prandtl number	
$Pr_{f(i)}$	Module i refrigerant liquid Prandtl number	
$Pr_{ref(i)}$	Module i refrigerant Prandtl number	
$p_{Rin(i)}$	Module i refrigerant inlet pressure	psi
$p_{Rout(i)}$	Module i refrigerant outlet pressure	psi
$Q_{mod(i)}$	Module i total heat transfer	Btu/hr
Re_{air}	Air side Reynolds number	
$Re_{eq(i)}$	Module i equivalent Reynolds number	
$Re_{f(i)}$	Module i liquid Reynolds number	
$Re_{g(i)}$	Module i vapor Reynolds number	

$Re_{ref(i)}$	Module i refrigerant Reynolds number	
$\mathrm{Res}(n, 2)$	Residual equation array for n modules	
$\rho_{avg(i)}$	Module i refrigerant average density	lbm/ft^3
$\rho_{f(i)}$	Module i refrigerant liquid density	lbm/ft^3
$\rho_{g(i)}$	Module i refrigerant vapor density	lbm/ft^3
$\rho_{ref(i)}$	Module i refrigerant density	lbm/ft^3
σ	Refrigerant surface tension	
τ_{wall}	Wall shear stress	
$T_{Ain(i)}$	Module i air inlet temperature	°F
$T_{air\ in\ back}$	Inlet air temperature to back modules of segment	°F
$T_{avg\ out\ front}$	Average air outlet temperature from front modules of segment	°F
$T_{Rin(i)}$	Module i refrigerant inlet temperature	°F
UA	Overall heat transfer coefficient	
μ_{air}	Air side viscosity	lbm/ft-hr
$\mu_{f(i)}$	Module i refrigerant liquid viscosity	lbm/ft-hr
$\mu_{g(i)}$	Module i refrigerant vapor viscosity	lbm/ft-hr
$\mu_{ref(i)}$	Module i refrigerant viscosity	lbm/ft-hr
$V_{(i)}$	Module i refrigerant velocity	ft/s
$\upsilon_{f(i)}$	Module i refrigerant liquid specific volume	ft^3/lbm
$\upsilon_{g(i)}$	Module i refrigerant vapor specific volume	ft^3/lbm
$\upsilon_{in(i)}$	Module i refrigerant inlet specific volume	ft^3/lbm
$\upsilon_{out(i)}$	Module i refrigerant outlet specific volume	ft^3/lbm
We_f	Refrigerant liquid Weber number	
W_g	Heat flux averaged void fraction	
$x_{out(i)}$	Module i refrigerant outlet quality	
$X_{tt(i)}$	Module i Lockhart-Martinelli correlating parameter	

1.0 INTRODUCTION

Accurate prediction of the mass in a system component is important for several reasons. In condenser modeling, accurate charge inventory prediction is required for simulating off design performance and transient behavior. In system modeling, the charge inventory of all components is required for the continuity equation. This work focuses on predicting refrigerant inventory in air cooled tube and fin condensers. In the single phase regions of the condenser, mass prediction is a straight forward calculation, while in the two-phase region, inventory is calculated with a void fraction correlation. The void fraction correlation provides a ratio of the tube cross sectional area occupied by the refrigerant vapor to the total cross sectional area for a given quality. There are many void fraction correlations available in the literature; Domanski and Didion [11], Baroczy [18], Zivi [17], Smith [19], Premoli et al. [12], Tandon et al. [10], and Hughmark [14]. The purpose of the work presented here, is to implement a number of these correlations into an existing condenser simulation program to predict the mass of the condenser and to provide a comparison of the correlations for different operating conditions. In addition, the factors most affecting the mass prediction are explored. Determination of the most accurate correlation requires experimental data of the charge for the condenser coil being modeled. This work will be available in the future.

The condenser simulation program used for this study was originally developed by Ragazzi [1]. The steady state simulation was designed to model air-cooled condensers which could be categorized as cross-flow heat exchangers with circular refrigerant tubes with air flowing over them. It is based on first principles and is general enough to use with condenser coils of

various geometries and for flows of different refrigerants. A complete description of the program and its adaptation to various condenser coil geometries is provided. The void fraction correlations to be incorporated into the simulation for the refrigerant mass prediction are discussed along with the integral solution techniques for the two phase mass equations.

The results indicate that the prediction of refrigerant inventory is heavily influenced by both the selection of void fraction correlation and the predicted length of the subcooling section in the condenser. This length is determined by the simulation program and is dependent upon the correlations used for the refrigerant side heat transfer coefficient, air side heat transfer coefficient and refrigerant pressure drop. It is observed that for mass determination it is as important for the simulation to properly predict the condenser outlet conditions, as it is to predict the overall capacity of the coil. In addition to available correlations, a parameter estimation process is used to improve the simulation's prediction of heat transfer, pressure drop and subcooling length. Inventory comparisons are provided for the best combination of heat transfer and pressure drop correlations.

In Chapter 2 , an overview of the Module Based Condenser Simulation program is provided. It outlines the simulation solution technique and the relevant heat transfer and pressure drop correlations. In addition, a discussion of the thermodynamic property routines used in the simulation is provided. In Chapter 3, a description of the condenser coil and modeling is given. A summary of the assumptions made is provided along with a detailed description of how the coil is modeled by the simulation. The void fraction correlations implemented in the program are discussed in Chapter 4, along with the various integral solution techniques. Chapter 5 is a discussion of the various parameter estimation techniques used to decrease the error in

the simulation program and in Chapter 6, the results are presented. They include the evaluation of the simulation performance along with the mass prediction results.

2.0 SIMULATION DESCRIPTION

2.1 Solution Technique

The Module Based Condenser Simulation program is a general, steady state, first principles condenser simulation which was developed by Ragazzi [1] to model air-cooled, cross-flow condenser coils of various geometries. The general technique used by the model is to divide the condenser coil into a user specified number of segments which in turn are divided into a number of modules. The inlet conditions to the segment are provided and then each module in the segment is treated as an individual heat exchanger and a Newton-Raphson solution is performed to determine the outlet conditions for each module. This procedure is used to analyze all the segments in the condenser coil. The governing equations for each module are the conservation of energy and momentum equations which provide a set of non-linear equations which are then solved with a Newton-Raphson iteration technique. A description of the Newton-Raphson technique is provided in Stoecker [2].

The operating conditions at the condenser coil inlet which must be specified for the simulation are the refrigerant inlet pressure, temperature and mass flow rate and the air inlet pressure, temperature, mass flow rate and relative humidity. These are what will be referred to as the inlet test conditions. All inlet conditions used for this analysis are experimental data points for which the actual condenser performance in known. As stated earlier, each module is treated as an individual heat exchanger so these same inlet variables must be specified for each module. Certain assumptions are made to get this information for each module, and hence, the accuracy of the solution is inherent on the validity of these assumptions. For example, the condenser coil modeled by Ragazzi consisted of two parallel, serpentine,

finned tubes with air flowing normal to the tubes. The inlet test conditions to the condenser coil were known (from experiments). The coil was divided into two segments, the front tube and the back tube. So the inlet test conditions on the refrigerant side for each segment were assumed to be the same as those for the inlet to the condenser, with the exception of the refrigerant mass flow rate which was assumed to be one-half of the total refrigerant mass flow rate. This is also the refrigerant mass flow rate for each module.

$$\dot{m}_{R\,mod} = \dot{m}_{Rseg} = 0.5 \times \dot{m}_{Rtot} \qquad [2.1]$$

For the air side inlet conditions, there is a slightly different situation. The inlet air temperature and pressure for each module, in the first tube of the coil modeled by Ragazzi, are equal to the experimental inlet pressure and temperature for the test point. The inlet air temperature to the second tube (segment) is set equal to the average outlet air temperature from the first tube (segment). This takes into account the heat transfer from the first segment. A more in depth discussion of this assumption is found in Ragazzi [1]. The air mass flow rate over each module must also be determined. For this example, it was assumed that the air flow rate in the duct was uniform and a weighted average was taken to determine the mass flow rate over each module.

$$\dot{m}_{A\,mod(i)} = \dot{m}_{Atot} \times \frac{L_{mod(i)}}{L_{seg}} \qquad [2.2]$$

Here the weighting function is a ratio of the length of the module to the total length of the segment, which is the total length of one tube. This weighting function represents a ratio of the air flow area for the module to the total air flow area and assumes that the refrigerant tubes are equally spaced. This weighting function can change depending on the geometry of the coil. For example, if a condenser coil had only one refrigerant tube, this tube could be

modeled as any number of segments. For this case, the total length of the condenser tube is used in the weighting function instead of the length of one segment. The subscript *(i)* in equation [2.2] designates the fact that this value will (or may) change for each module. This notation is utilized throughout this thesis. For the refrigerant module mass flow rate, the value is the same for all modules in a segment.

One additional determination needs to be made for the condenser simulation, whether the user wants to fix the length of a module or the outlet quality of the module. The example above would work well with either of these versions. The module length could be specified for each module and the program will solve for the outlet quality. If the outlet quality is specified for each module then the program determines the length of each module required to obtain this condition. Both versions have their advantages and disadvantages. For example, the fixed quality version is useful if the length of each of the condenser regions (superheated vapor, two phase and subcooled liquid) is to be determined. For more complicated geometry, however, the fixed length version is more adaptable to the coil geometry as will be shown later. One source of error in the fixed length version is a result of the determination of the local refrigerant side heat transfer coefficients. The heat transfer correlation used for a module is always based on the outlet conditions. For example, if the inlet condition of a module is superheated vapor and the outlet condition of the module is two phase, then the program uses the two phase correlation to determine the heat transfer coefficient for the entire module. In reality, the transition from superheated vapor to two phase refrigerant occurred somewhere in the module, but the fixed length version of the program is unable to determine this transition point. This problem only occurs at the two transition points (superheated vapor to two

phase refrigerant, and two phase refrigerant to subcooled liquid), and can be compensated for somewhat by making the length of the modules smaller.

As stated previously, the simulation uses a Newton-Raphson solution technique for each segment to determine the outlet conditions of each module. The Newton-Raphson variables for the fixed length version of the simulation are the module outlet enthalpy ($h_{Rout(i)}$) and the module outlet pressure ($p_{Rout(i)}$). The residuals equations for the Newton-Raphson iteration are given by,

$$\mathrm{Res}(i,1) = h_{Rout(i)} - h_{Rin(i)} - \frac{Q_{mod(i)}}{\dot{m}_{R\ mod}} = h_{Rout(i)} - h_{Rout(i-1)} - \frac{Q_{mod(i)}}{\dot{m}_{R\ mod}} \qquad [2.3]$$

$$\mathrm{Res}(i,2) = p_{Rout(i)} - p_{Rin(i)} - \Delta P_{mod(i)} = p_{Rout(i)} - p_{Rout(i-1)} - \Delta P_{mod(i)} \qquad [2.4]$$

In equation [2.3], $Q_{mod(i)}$ represents the total heat transfer from the module and in equation [2.4] $\Delta P_{mod(i)}$ represents the total pressure drop in the module. The module heat transfer is calculated using an effectiveness-NTU method and the total pressure drop is calculated using a pressure drop correlation.

For certain coil geometries, an additional Newton Raphson variable is used. This variable is the average outlet air temperature from the modules. This situation occurs when the refrigerant tube is wrapped from the front of the coil to the back of the coil with a return bend and the refrigerant inlet to the tube is located in the back. In this situation, the portion of the tube at the back of the condenser requires the average air outlet temperature from the portion of the tube in the front of the coil to use as its inlet air condition. This is accomplished by modeling the tube as one segment and having an equal number of modules in both the front and rear portions of the tube. The Newton-Raphson residual equation which is added to the iteration is given by,

$$\mathrm{Res} = T_{air\ in\ back} - T_{avg\ out\ front} \qquad [2.5]$$

A more in depth discussion of this procedure is provided in Ragazzi [24].

2.2 Effectiveness-NTU Method

The effectiveness-NTU method is used in the condenser simulation to determine the heat rejected by each module ($Q_{mod(i)}$). The general form of the equation is,

$$Q_{mod(i)} = \varepsilon_{(i)} C_{min(i)} (T_{R\,in(i)} - T_{A\,in(i)}) \quad [2.6]$$

The module effectiveness $\varepsilon_{(i)}$ is defined as the ratio of the actual heat transfer rate to the "ideal" or maximum heat transfer rate. This value is dependent on the ratio of heat capacity rates and the number of transfer units (NTU),

$$C_{ratio(i)} = \frac{C_{min(i)}}{C_{max(i)}} \quad [2.7]$$

$$\text{NTU} = \frac{\text{UA}}{C_{min(i)}} \quad [2.8]$$

The heat capacity rates are calculated for the both the refrigerant and the air, and the minimum value of these is $C_{min(i)}$ and the maximum value is designated as $C_{max(i)}$. The heat capacity rate is defined as the mass flow rate times the specific heat,

$$C_{ref(i)} = \dot{m}_{R\,mod} \times c_{pR(i)} \quad [2.9]$$

$$C_{air(i)} = \dot{m}_{A\,mod(i)} \times c_{pA(i)} \quad [2.10]$$

The variable UA in equation [2.8] is the overall heat transfer coefficient and is the sum of the convective resistances on the air side and refrigerant side of the condenser coil and the conductive resistance of the tube wall.

$$\text{UA} = \frac{1}{\frac{1}{\eta_s h_{air(i)} A_{so(i)}} + \frac{t}{k_{tube} A_{mean}} + \frac{1}{h_{ref(i)} A_{si(i)}}} \quad [2.11]$$

The effectiveness is then calculated for one of three cases:

1. $C_{min(i)} = C_{air(i)}$ and the refrigerant is single phase

$$\varepsilon_{(i)} = \frac{1}{C_{ratio(i)}} \left[1 - e^{-\left[1 - e^{-\text{NTU}}\right] \times C_{ratio(i)}} \right] \quad [2.12]$$

2. $C_{min(i)} = C_{ref(i)}$ and the refrigerant is single phase

$$\varepsilon_{(i)} = e^{-\left[1-e^{-NTU \times C_{ratio(i)}}\right]/C_{ratio(i)}} \qquad [2.13]$$

3. Refrigerant is two phase: $\frac{C_{min(i)}}{C_{max(i)}} \rightarrow 0$

$$\varepsilon_{(i)} = 1 - e^{-NTU} \qquad [2.14]$$

The third case is the limiting case which occurs in the condensing region when the refrigerant side specific heat approaches infinity. Once the coil geometry and material are specified, equation [2.11] indicates that there are three unknowns in the equation which must be determined: η_s, $h_{air(i)}$ and $h_{ref(i)}$. The specific formulation for these variables is provided through heat transfer coefficient correlations which are found in the following sections.

2.3 Air Side Heat Transfer

Two correlation forms are available in the simulation program for determining the air side heat transfer coefficient. They are the Colburn j-factor [13] and the Temperature Limit Principle [24]. Both correlations provide an overall air side heat transfer coefficient which is dependent on the inlet air conditions. The module heat transfer coefficient is then determined by applying an appropriate weighting function. Both methods assume a uniform mass flow rate over the coil.

2.3a Colburn j-factor

The Colburn j-factor is determined for the condenser coil using experimental data and a modified Wilson plot technique [3]. An advantage of this technique is that the surface efficiency (η_s) of the coil is found in addition to the j-factor correlation. Another advantage is that if no experimental data

are available, Kays and London [13] have tabulated data for various heat exchanger geometries which can be used to determine this correlation. The general formulation for the Colburn j-factor is,

$$j = \frac{Nu_{air}}{Re_{air}\, Pr_{air}^{1/3}} \qquad [2.15]$$

where,

$$Nu_{air} = \text{Air Nusselt Number} = \frac{h_{air} D_h}{k_{air}} \qquad [2.16]$$

$$Re_{air} = \text{Air Reynolds Number} = \frac{G_{air} D_h}{\mu_{air}} \qquad [2.17]$$

$$Pr_{air} = \text{Air Prandtl Number} = \frac{\mu_{air} c_{p\,air}}{k_{air}} \qquad [2.18]$$

Once the j-factor has been determined for various data points using the modified Wilson plot technique, a correlation is found as a function of the air side Reynolds number. A least squares fit was utilized in the determination of this correlation, and for the coil used in this thesis the correlation was found to be,

$$j = 0.166\, Re_{air}^{-0.4} \qquad [2.19]$$

The heat transfer coefficient correlation is then found by rearranging equation [2.15] and canceling terms.

$$h_{air} = \frac{G_{air}\, c_{p\,air}\, j}{Pr_{air}^{2/3}} \qquad [2.20]$$

This is the overall air side heat transfer coefficient. The module heat transfer coefficient is found by multiplying the overall coefficient by a weighting function. The weighting function used by Ragazzi [1] is given by,

$$h_{air(i)} = h_{air} \times \frac{L_{mod(i)}}{L_{seg}} \qquad [2.21]$$

This is the same weighting function used to determine the module air mass flow rate in equation [2.2]. A more in depth discussion of the modified Wilson plot technique can be found in Weber [4].

2.3b Temperature Limit Principle

The Temperature Limit Principle is a method developed by Marin [24] to determine the air side heat transfer coefficient. The basis for this method is the following experimental observation. It was observed on the experimental apparatus, that in order to maintain constant refrigerant inlet properties (i.e. pressure, temperature and mass flow rate) while changing the inlet air temperature, a corresponding change in the air mass flow rate was required. In addition, it was observed that when the air mass flow was changed, the outlet air temperature remained constant. From this observation a method for determining the air side heat transfer correlation was developed. The advantage with this method is that only a small number of data points are required to develop the correlation. The correlation for the coil used in this thesis was found by Marin to be,

$$Nu_{air} = 0.055\, Re_{air}^{0.918} Pr_{air}^{0.4} \qquad [2.22]$$

$$h_{air} = \frac{k_{air} Nu_{air}}{D_h} \qquad [2.23]$$

This is the overall air side heat transfer coefficient. The module air side heat transfer coefficient is still given by equation [2.21].

2.4 Refrigerant Side Heat Transfer

The calculation of the local refrigerant side heat transfer coefficient ($h_{ref(i)}$) is dependent upon the refrigerant phase. If the refrigerant in the module is either a superheated vapor or a subcooled liquid, a single phase correlation is used. If the module contains a two phase flow, a two phase correlation is used. In Section 2.3, the module air side heat transfer coefficient was found by determining an overall air side coefficient for the coil and applying a weighting function. On the refrigerant side, this procedure is not

necessary. There are many heat transfer coefficient correlations available in the literature for single and two phase flows through circular cross section tubes. This allows the heat transfer coefficient for each module to be calculated independently. This section provides a discussion of the correlations available in the simulation which are relevant to this work.

2.4a Single Phase Correlation

In the single phase region of the condenser, the Dittus-Boelter [20] correlation is used to determine the refrigerant heat transfer coefficient for the module. This correlation is a general empirical correlation for single phase flow through cylindrical tubes. It expresses the dimensionless Nusselt number as a function of the dimensionless Reynolds and Prandtl numbers. The general form of the correlation for cooling is given by,

$$Nu_{ref(i)} = 0.023 \times Re_{ref(i)}^{0.8} \times Pr_{ref(i)}^{0.3} \qquad [2.24]$$

The heat transfer coefficient for the module is related to the Nusselt number by,

$$h_{ref(i)} = \frac{Nu_{ref(i)} \times k_{ref(i)}}{D_i} \qquad [2.25]$$

The dimensionless Reynolds number and Prandtl number for the module are given by,

$$Re_{ref(i)} = \frac{\dot{m}_{R\,mod} \times D_i}{A_i \times \mu_{ref(i)}} \qquad [2.26]$$

$$Pr_{ref(i)} = \frac{\mu_{ref(i)} \times c_{pR(i)}}{k_{ref(i)}} \qquad [2.27]$$

It should be noted that additional single phase correlations are available for use in the simulation program and can be found in Ragazzi [1].

2.4b Two Phase Correlations

In the calculation of the condenser mass inventory, three refrigerant two phase correlations are compared, the Cavallini-Zecchin [5], Shah [25] and Dobson [6]. All three correlations were developed for annular flow regimes. The Cavallini-Zecchin and Shah correlations are examples of the more general empirical correlations available and were both developed with a large amount of data from various fluids. The Dobson correlation, on the other hand, was developed specifically for R134a. Overall, the results presented in this paper found that the Dobson correlation provided more accurate heat transfer results. This is discussed in greater detail in Chapter 6.

The general form of the Cavallini-Zecchin correlation is given by,

$$h_{ref(i)} = 0.05 \times Re_{eq(i)}^{0.8} \times Pr_{f(i)}^{0.33} \left[\frac{k_{f(i)}}{D_i} \right] \qquad [2.28]$$

where,

$$Re_{eq(i)} = Re_{f(i)} + Re_{g(i)} \left[\frac{\mu_{g(i)}}{\mu_{f(i)}} \right] \left[\frac{\rho_{f(i)}}{\rho_{g(i)}} \right]^{0.5} \qquad [2.29]$$

$$Re_{f(i)} = \frac{G_{ref} \times D_i}{\mu_{f(i)}} \left(1 - x_{out(i)}\right) \qquad [2.30]$$

$$Re_{g(i)} = \frac{G_{ref} \times D_i}{\mu_{g(i)}} \left(x_{out(i)}\right) \qquad [2.31]$$

$$Pr_{f(i)} = \frac{\mu_{f(i)} \times c_{pf(i)}}{k_{f(i)}} \qquad [2.32]$$

The use of this correlation is recommended for equivalent Reynolds numbers in the range of $7,000 < Re_{eq(i)} < 53,000$.

The Dobson correlation which was developed for R134a is a function of the Lockhart-Martinelli correlating parameter X_{tt}. Its general form is given by,

$$h_{ref(i)} = 2.61 \times h_{f(i)} \; / \; X_{tt(i)}^{0.80} \qquad [2.33]$$

where,

$$X_{tt(i)} = \left[\frac{\rho_{g(i)}}{\rho_{f(i)}}\right]^{0.5}\left[\frac{\mu_{f(i)}}{\mu_{g(i)}}\right]^{0.125}\left[\frac{1 - x_{out(i)}}{x_{out(i)}}\right]^{0.875} \quad [2.34]$$

and $h_{f(i)}$ is the liquid heat transfer correlation,

$$h_{f(i)} = \frac{0.023 \times k_{f(i)} \times Re_{f(i)}^{0.8} \times Pr_{f(i)}^{0.3}}{D_i} \quad [2.35]$$

Equation [2.35], is the Dittus-Boelter single phase liquid heat transfer coefficient and the liquid Reynolds number and Prandtl number are given by equations [2.30] and [2.32]. The use of this correlation is recommended for qualities greater than 0.05 and the single phase correlation is used for qualities below 0.05.

The Shah two phase correlation is also a function of the Dittus-Boelter single phase liquid heat transfer coefficient as given by equation [2.35]. It's general form is,

$$h_{ref(i)} = \Psi\, h_{1(i)} \quad [2.36]$$

where,

$$\Psi = 1 + \frac{3.8}{Z^{0.95}} \quad [2.37]$$

$$Z = \left(\frac{1}{x_{in(i)}} - 1\right)^{0.8} p_{red(i)}^{0.4} \quad [2.38]$$

$$p_{red(i)} = \frac{p_{sat(i)}}{p_{crit}} \quad [2.39]$$

$$h_{1(i)} = h_{f(i)}\left(1 - x_{in(i)}\right)^{0.8} \quad [2.40]$$

and $h_{f(i)}$ is given by equation [2.35]. The applicability of this correlation is recommended for liquid Reynolds numbers in the range of $350 < Re_{f(i)} < 35,000$.

2.5 Refrigerant Pressure Drop

As is the case for the refrigerant heat transfer coefficient, the refrigerant pressure drop across the module is dependent on the phase of the refrigerant. The simulation program contains both single phase and two phase pressure drop correlations. There are three components for the module pressure drop: frictional pressure drop, momentum pressure drop and gravitational pressure drop.

$$\Delta P_{mod(i)} = \Delta P_{fric(i)} + \Delta P_{mom(i)} + \Delta P_{grav(i)} \qquad [2.41]$$

The frictional pressure drop and the momentum pressure drop are determined for each module individually. For the gravitational pressure drop, the total elevation change in the condenser is used to determine an overall gravitational pressure drop which is then multiplied by the weighting function $L_{mod(i)} / L_{cond}$ to provide the gravitational pressure drop for each module. This weighting function uses the total length of refrigerant tube. The frictional pressure drop in the single phase region is found using the Fanning friction factor. In the two phase region, there is a choice of correlations, the Lockhart-Martinelli [22] and Souza [26]. The momentum pressure drop is obtained through a control volume analysis over the module [1].

2.5a Single Phase Correlation

The frictional pressure drop in the single phase regions of the condenser coil is determined using the Fanning friction factor which is given by,

$$f = \frac{\tau_{wall}}{0.5\rho_{ref(i)}V_{(i)}^{2}} \qquad [2.42]$$

For laminar flow the following correlation can be used,

$$f = 16 / Re_{ref(i)} \qquad [2.43]$$

For turbulent refrigerant flow,

$$f = 0.046 \times Re_{g(i)}^{-0.2} \qquad \text{for vapor} \qquad [2.44]$$

or,

$$f = 0.079 \times Re_{f(i)}^{-0.25} \qquad \text{for liquid} \qquad [2.45]$$

The frictional pressure drop for the module is then given by,

$$\Delta P_{fric(i)} = \frac{2 \times f \times G_{ref}^{\ 2} \times L_{mod(i)}}{D_i \times \rho_{ref(i)} \times UC} \qquad [2.46]$$

where UC is a units conversion given by,

$$UC = g_c \times 3600 \frac{s}{hr} \times 3600 \frac{s}{hr} \times 144 \frac{in^2}{ft^2} \qquad [2.47]$$

The momentum pressure drop for the module is given by,

$$\Delta P_{mom(i)} = \frac{-G_{ref}^{\ 2}\left(v_{out(i)} - v_{in(i)}\right)}{UC} \qquad [2.48]$$

The gravitational pressure drop for the module is given by,

$$\Delta P_{grav(i)} = \rho_{ref(i)} g_c H \left[\frac{L_{mod(i)}}{L_{cond}}\right] \qquad [2.49]$$

2.5b Two Phase Correlations

The simulation program allows for the choice of two frictional pressure drop correlations in the two phase region. They are the Lockhart-Martinelli correlation [22] and the Souza [26] correlation. The equations given for the momentum and gravitational pressure drop in the two phase region are not affected by the choice of frictional pressure drop correlation.

The Lockhart-Martinelli frictional pressure drop correlation uses a separated flow model which was developed by Lockhart and Martinelli based on their studies of air-water flows. The general form of this frictional two phase pressure drop correlation is given by,

$$\Delta P_{fric(i)} = \Phi_{vap}^{\ 2} \left[\frac{2 \times f_{vap} \times G_{ref}^{\ 2} \times x_{out(i)}^{\ 2} \times L_{mod(i)}}{D_i \times \rho_{g(i)} \times UC}\right] \qquad [2.50]$$

where Φ is a two phase frictional multiplier which is dependent on whether there is laminar or turbulent flow in the liquid and vapor, and f_{vap} is the vapor Fanning friction factor given by equation [2.44]. Various forms for Φ are available in the literature, Soliman et al. [15] and Chisholm [16]. The two-phase frictional multiplier developed by Soliman et al. is for the case of both turbulent liquid and vapor flow (the turbulent-turbulent case) and is given by,

$$\Phi_{vap} = 1 + 2.85\, X_{tt(i)}^{0.523} \qquad [2.51]$$

where $X_{tt(i)}$ is the Lockhart-Martinelli parameter given by equation [2.34].

The Souza two phase frictional pressure drop correlation was developed for use with refrigerants, specifically R134a. It is a function of both the Lockhart-Martinelli correlating parameter and the Froude number. It's general form is given by,

$$\Delta P_{fric(i)} = \frac{dPlo \times \Phi}{UC} \qquad [2.52]$$

where,

$$dPlo = \frac{2.0 \times 0.079 \times L_{mod(i)} \times G^{2.0}}{\rho_{f(i)} \times D_i \times Re_{f(i)}^{0.25}} \qquad [2.53]$$

$$\Phi = \left[1.376 + C1\left(X_{tt(i)}^{-C2}\right)\right](1 - x)^{1.75} \qquad [2.54]$$

$$C1 = 7.242 \quad \text{and} \quad C2 = 1.655 \qquad Fr > 0.7 \qquad [2.55]$$

$$C1 = 4.172 + 5.480\,Fr - 1.564\,Fr^{2.0} \qquad Fr < 0.7 \qquad [2.56]$$
$$C2 = 1.773 - 0.169\,Fr$$

$$Fr = \frac{G^{2.0}}{D_i \times 32.2 \times \rho_{f(i)}} \qquad [2.57]$$

In the simulation program an average value for Φ at the inlet and outlet of the module is taken.

The momentum pressure drop is found by applying a momentum

balance to the module which yields,

$$\Delta P_{mom(i)} = -\frac{G_{ref}^2}{UC}\left\{\left[\frac{x^2}{\rho_g \alpha} + \frac{(1-x)^2}{\rho_f(1-\alpha)}\right]_{out(i)} - \left[\frac{x^2}{\rho_g \alpha} + \frac{(1-x)^2}{\rho_f(1-\alpha)}\right]_{in(i)}\right\} \quad [2.58]$$

where α is the void fraction correlation proposed by Zivi [17] and is given by,

$$\alpha = \frac{1}{1 + \left[\frac{1-x}{x}\right]\left[\frac{\rho_{g(i)}}{\rho_{f(i)}}\right]^{2/3}} \quad [2.59]$$

The gravitational pressure drop is calculated based on a homogeneous flow model and has the same general form as the single phase pressure drop except an average density is used.

$$\Delta P_{grav(i)} = \rho_{avg(i)} g_c H\left[\frac{L_{mod(i)}}{L_{cond}}\right] \quad [2.60]$$

where,

$$\rho_{avg(i)} = \left[\frac{1}{v_{f(i)} + x_{(i)}\left(v_{g(i)} - v_{f(i)}\right)}\right] \quad [2.61]$$

2.5c Pipe Component Pressure Drop

In addition to the pressure drop in the modules of the condenser, there is also a pressure drop in various components of the condenser coil such as the manifolds and return bends. A correlation for determining the pressure drop in these components was developed by Paliwoda [23]. He developed a generalized method for determining the pressure drop across pipe components with two-phase flow. This includes, manifolds, return bends, T-junctions, valves, etc. The general component pressure drop correlation for the condenser is given by,

$$\Delta P_{com} = \frac{G_{ref}^2 \times \xi \times \beta_c}{2 \times \rho_g \times UC} \quad [2.62]$$

For this condenser's manifold,

$$\xi = 2.7 \quad [2.63]$$

and,

$$\beta_c = [\theta + 0.58x(1-\theta)][1-x]^{0.333} + x^{2.276} \quad [2.64]$$

where,

$$\theta = \frac{\rho_g}{\rho_f}\left[\frac{\mu_f}{\mu_g}\right]^{0.25} \quad [2.65]$$

For the return bends,

$$\xi = 0.12 \quad [2.66]$$

and,

$$\beta_c = [\theta + 3.0x(1-\theta)][1-x]^{0.333} + x^{2.276} \quad [2.67]$$

Overall it was found that the frictional pressure drop in the return bends is of the same order of magnitude as was found for the module frictional pressure drop. The manifold frictional pressure drop, on the other hand, was quite large in comparison to the module pressure drop. The addition of these added pressure drop calculations helped to improve the overall pressure drop prediction of the simulation program.

2.6 Flow Regime

One additional aspect of the simulation program is to determine the flow regime of the two phase refrigerant. The correlations used to determine the local two phase heat transfer coefficients are all based on data for annular flow regimes. It is desired to determine the flow regime for each module to verify the validity of the annular flow correlations. Rahman et al. [27] provide a method for determining the flow regime based on the Weber and Froude numbers. The Weber number correlations are given by,

$$We = 0.85\, Re_{g(i)}^{0.79}\left(\frac{\mu_{g(i)}^{2.0}}{\rho_{g(i)}\,\sigma\, D_i}\right)^{0.3}\left[\left(\frac{\mu_{g(i)}}{\mu_{f(i)}}\right)^2\left(\frac{\rho_{f(i)}}{\rho_{g(i)}}\right)\right]^{0.084}\left(\frac{X_{tt(i)}}{\Phi_{g(i)}^{2.55}}\right)^{0.157} \quad [2.68]$$

for $Re_{f(i)} > 1250$ and,

$$We = 2.45\, Re_{g(i)}^{0.64} \left(\frac{\mu_{g(i)}^2}{\rho_{g(i)}\, \sigma\, D_i} \right)^{0.3} \left(\frac{1}{\Phi_{g(i)}^{0.4}} \right) \qquad [2.69]$$

for $Re_{f(i)} \leq 1250$ where,

$$\Phi_{g(i)} = 1 + 1.09\, X_{tt(i)}^{0.039} \qquad [2.70]$$

The Froude number is given by,

$$Fr = 0.025\, Re_{f(i)}^{1.59} \left(\frac{\Phi_{g(i)}}{X_{tt(i)}} \right)^{1.5} \left(\frac{1}{Ga^{0.5}} \right) \qquad Re_{f(i)} \leq 1250 \qquad [2.71]$$

$$Fr = 1.26\, Re_{f(i)}^{1.04} \left(\frac{\Phi_{g(i)}}{X_{tt(i)}} \right)^{1.5} \left(\frac{1}{Ga^{0.5}} \right) \qquad Re_{f(i)} > 1250 \qquad [2.72]$$

where,

$$Ga = \frac{g_c\, D_i^3}{\nu_{f(i)}^2} \qquad [2.73]$$

With these definitions the following criteria were set by Rahman to determine the flow regime. For Weber numbers greater than 30, the flow regime was considered to be mist flow. Weber numbers between 20 and 30 corresponded to the transition between mist and annular flow. For Weber numbers less than 20, the flow regime is considered to be annular for Froude numbers greater than 7 and wavy for Froude numbers less than 7. Froude numbers equal to 7 with a Weber number less than 20 corresponded to the transition point between the annular and wavy flow regimes.

2.7 Refrigerant Thermodynamic Property Routines

The thermodynamic property subroutines utilized by the simulation were developed by the National Institute of Standards and Technology (NIST) [7]. The use of these subroutines allows for greater flexibility in the model. There is a choice of two equations of state, the Benedict-Webb-Rubin (BWR) or the Carnahan-Starling-DeSantis (CSD) equation of state. In addition, there are coefficients for the CSD equation of state for up to 26 refrigerants and mixtures of up to five components can be analyzed. The BWR subroutines are more accurate, especially for R134a which is the reference refrigerant [8]. When using the Benedict-Webb-Rubin equation of state, properties for other refrigerants are found using the principle of corresponding states. The CSD equation of state subroutines were chosen for use in the simulation program because they allowed the flexibility of looking at mixtures.

A comparison of the enthalpy change and saturation temperatures for the two equations of state for R134a was made to determine the error that could result through the use of the less accurate CSD equation of state. It was found that there were errors in the saturation temperatures of less than 0.5 %. For the latent heat, it was found that the error in the CSD equation of state (EOS) ranged from 3.4% to 4.7% for a saturation pressure of 150 psi to 250 psi which is shown in Figure 2.1.

The experimental data taken for the condenser coil is also analyzed with the CSD equation of state subroutines, so this error should not affect the validation of the simulation. An interface program was written for use with the FORTRAN subroutines. This interface allows the subroutines to be implemented into existing programs with relatively few problems.

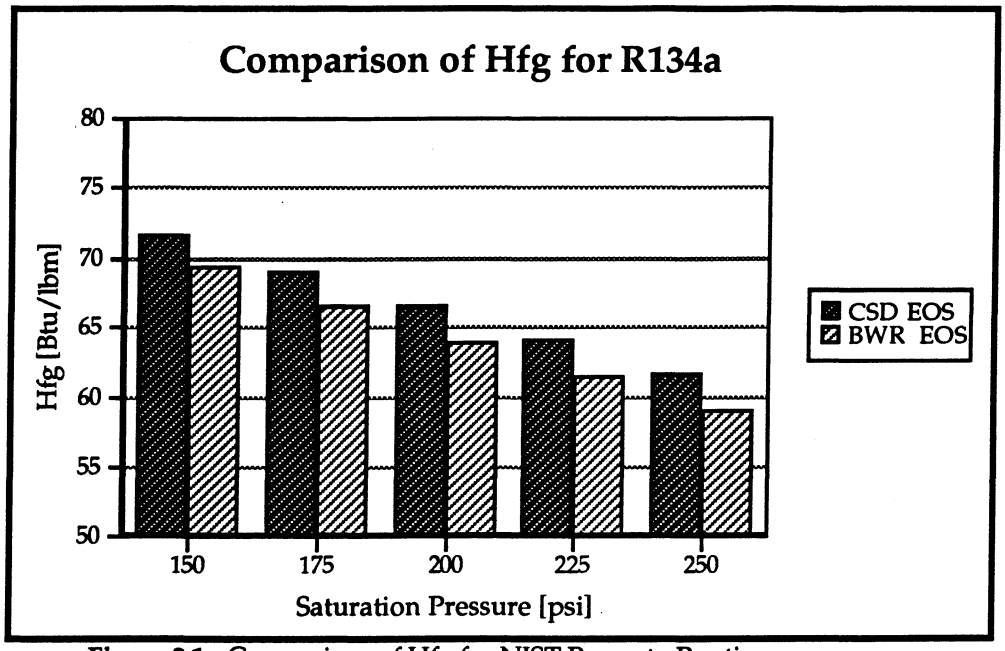

Figure 2.1 - Comparison of Hfg for NIST Property Routines

3.0 COIL GEOMETRY AND MODELING

The condenser coil used for this analysis is a cross-flow tube and fin heat exchanger with a fairly complex geometry. This chapter provides an overview of the coil geometry along with a discussion of the assumptions made in the computer modeling.

The inlet to the condenser coil is a manifold which is divided into a number of sections where the refrigerant flows in and out of the refrigerant tubes. In each of these sections, a number of tubes circulate the refrigerant through the width of the condenser and return to the manifold where the refrigerant drops into the next section of tubes. Figure 3.1 below, is a top view of the condenser coil which shows the manifold inlet and the top tube configuration.

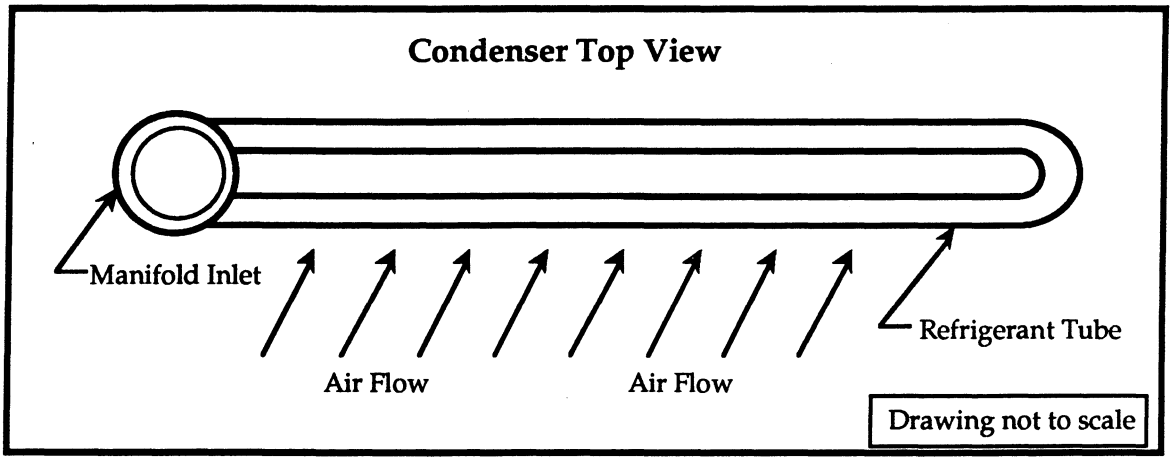

Figure 3.1 - Condenser Top View

Figure 3.2 shows the number of sections in the condenser coil and the number of refrigerant tubes in each section. It also indicates the air flow direction and whether the refrigerant inlet is in the front of the tube or the rear of tube. This is an important point in the condenser modeling. In Section 2.1 the modeling of a refrigerant tube whose air inlet was in the front and refrigerant inlet was in the rear was discussed. This situation requires the additional Newton-Raphson variable which determined the average air

outlet temperature from the front of the refrigerant tube to use as the air inlet temperature to the back of the refrigerant tube. This will greatly affect the number of segments required to model this coil.

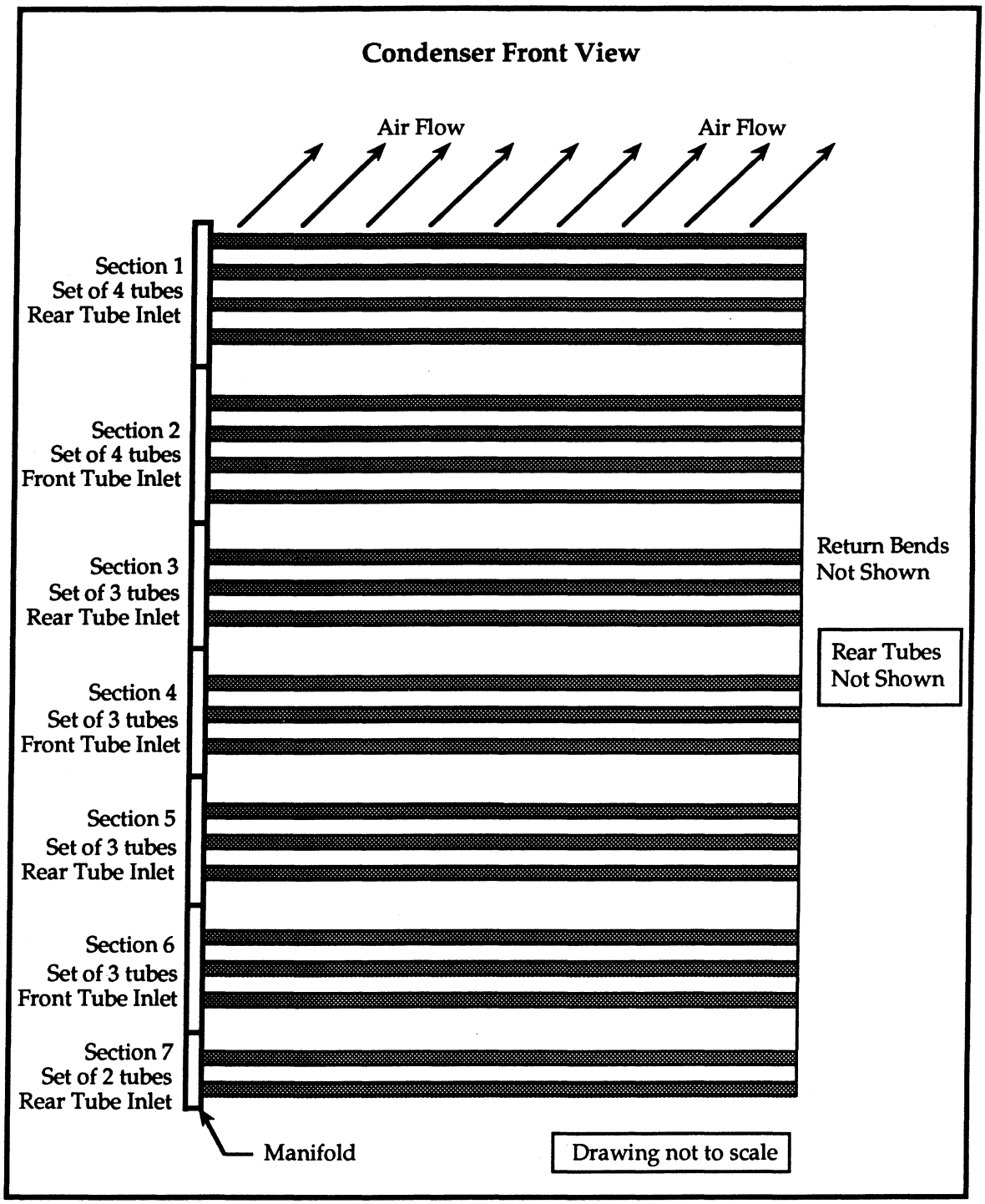

Figure 3.2 - Condenser Front View

It should be noted, that the distance between manifold sections in Figure 3.2 has been exaggerated to enhance it's clarity. This complex geometry requires the use of the fixed length version of the simulation program. The fixed length version allows the sections of the condenser which need the additional Newton-Raphson variable to be analyzed. For this coil geometry, the condenser is divided into ten segments. The manifold sections which have a rear inlet are modeled as one segment (sections 1, 3, 5 and 7), alternatively, the sections with a frontal air inlet are modeled as two segments (sections 2, 4 and 6).

As stated previously the coil is modeled by ten segments. It is assumed that the inlet conditions to all of the tubes in each manifold section are equivalent. This allows the use of only one tube in the Newton-Raphson solution. Table 3.1 below shows the number of modules each segment was divided into for the inventory analysis presented in Chapter 6.

Number of Modules per Segment for Inventory Analysis		
Manifold Section	**Segment Number**	**Number of Modules**
1	1	4
2	2	2
	3	2
3	4	4
4	5	2
	6	2
5	7	4
6	8	2
	9	2
7	10	4

Table 3.1 - Number of Modules Per Segment

Table 3.2 gives some of the overall condenser dimensions needed to run the simulation.

Condenser Dimensions		
Variable	**Value**	**Units**
Condenser Total Tube Length	81.4	ft
Condenser Frontal Tube Length	40.7	ft
Inside Tube Diameter	0.204	in
Outside Tube Diameter	0.235	in

Table 3.2 - Condenser Dimensions

Here the total condenser tube length is used in the weighting function to determine the gravitational pressure drop in the module given by equations [2.49] and [2.60]. For the coil modeled by Ragazzi, the frontal condenser tube length was used to determine the air flow rate over a module given by equation [2.2] and the air side heat transfer coefficient in equation [2.21]. For this coil, the frontal length is used to determine the air mass flow rate because this represents the ratio of the area which the module air flow occupies to the total air flow area assuming the refrigerant tubes are equally spaced. For the heat transfer coefficient, the total length is used because the overall air side heat transfer coefficient found is for the whole condenser, so the ratio of the length of a module to the total condenser length better represents the portion of the total coefficient.

Certain assumptions are made to determine the refrigerant properties along the condenser length. The first assumption made is that refrigerant mass flow rate entering the tubes from the manifold is uniformly divided among the number of tubes giving,

$$\dot{m}_{R\,mod} = \frac{\dot{m}_{Rtot}}{n_{tubes}} \qquad [3.1]$$

In addition, the inlet conditions (pressure, temperature and enthalpy) to each tube in the manifold section are assumed to be equal. The return bends and

manifolds are assumed to be adiabatic since they are not exposed to the airflow, however, the pressure drop across the return bends and manifolds is included in the simulation.

For example, the inlet pressure and temperature to the condenser is specified by the user. The program determines the condenser inlet enthalpy and the pressure drop in the first manifold. So the inlet conditions to the first segment are the enthalpy at the inlet of the condenser and the inlet pressure minus the manifold pressure drop. With these two thermodynamic properties set, all the additional segment inlet conditions are found using the thermodynamic property routines. The simulation then solves the Newton-Raphson problem for the first segment which provides the outlet conditions of the first segment. To determine the inlet conditions for the second segment, the inlet enthalpy is equal to the outlet enthalpy of the first segment (the manifold is adiabatic). The inlet pressure is the outlet pressure of the first segment minus the manifold pressure drop. Return bend pressure drops are calculated only in manifold sections which are modeled with two segments.

This procedure continues until all segments of the condenser have been analyzed. The total mass of the condenser is given by,

$$m_{total} = \sum_{m=1}^{p} \left[n_{tubes(m)} \sum_{i=1}^{n} m_{mod(i)} \right] \qquad [3.2]$$

If there were only one refrigerant tube per segment then the total mass of the condenser is given by,

$$m_{total} = \sum_{i=1}^{n} m_{mod(i)} \qquad [3.3]$$

Since there are more tubes per segment in this coil, the total mass of the condenser is given by,

$$m_{total} = \sum_{i=1}^{n} m_{mod(i)} \times n_{tubes} \qquad [3.4]$$

4.0 PREDICTION OF INVENTORY

The simulation program solves for the outlet conditions (temperature, pressure, quality, enthalpy, etc.) of each module. It is these values which are needed to determine the refrigerant mass in each module using a void fraction correlation. Five void fraction correlations are added to the simulation for comparison purposes. They are the homogeneous void fraction model [21], the Domanski and Didion correlation [11], the Tandon et al. correlation [10], the Premoli et al. correlation [12] and the Hughmark correlation [14] . Section 4.1 provides an overview of the general governing equations needed to calculate the refrigerant inventory. In Section 4.2 the specific format for each of the void fraction models is presented and the different integral solution techniques are discussed in Section 4.3.

4.1 General Equations

The general refrigerant inventory equations are provided for both the single phase regions and two phase region of the condenser. In the single phase regions, the condenser mass is a function of the density and volume of the module only. In the two phase region, the mass is also a function of the quality which leads to a more complicated integral equation. The mass of each module is determined using its inlet and outlet conditions, and then the total condenser mass is found by summing all the module masses.

$$m_{total} = \sum_{i=1}^{n} m_{mod(i)} \qquad [4.0]$$

There are many void fraction correlations are available in the literature and Rice [9] provides an excellent summary of some of them.

4.1a Single Phase Region

In the single phase region of the condenser, the refrigerant mass is a function of the density and module volume and is given by,

$$m_{mod(i)} = \int_0^L A_i \, \rho_{(i)} \cdot dl \qquad [4.1]$$

where $L = L_{mod(i)}$. Since the cross sectional area of the condenser is constant (the tube inside diameter is constant), this equation can be represented by,

$$m_{mod(i)} = V_{mod(i)} \times \rho_{avg(i)} = A_i \times L_{mod(i)} \times \rho_{avg(i)} \qquad [4.2]$$

where,

$$\rho_{avg(i)} = \frac{\rho_{in(i)} + \rho_{out(i)}}{2} \qquad [4.3]$$

4.1b Two Phase Region

In the two phase region of the condenser, the total mass of the module can be found by adding the mass of the liquid and the mass of the vapor together.

$$m_{mod(i)} = m_{vap(i)} + m_{liq(i)} \qquad [4.4]$$

To determine the mass of the vapor and liquid in the module the same integral that was used in equation [4.1] is performed for the vapor and liquid refrigerant separately. The mass of the vapor present in the module is given by,

$$m_{vap(i)} = \int_0^L \rho_g A_g \cdot dl \qquad [4.5]$$

and the mass of the liquid present in the module is given by,

$$m_{liq(i)} = \int_0^L \rho_f \, A_f \cdot dl \qquad [4.6]$$

The void fraction, alpha, is defined as the ratio of the cross sectional area occupied by the vapor to the total tube cross sectional area, $\alpha = A_g \, / \, A_i$. Equations [4.5] and [4.6] can then be written in the following form,

$$m_{vap(i)} = \rho_g \, A_i \int_0^L \alpha \cdot dl \qquad [4.7]$$

and,

$$m_{liq(i)} = \rho_f A_i \int_0^L (1 - \alpha) \cdot dl \qquad [4.8]$$

The total refrigerant mass in the module is then given by,

$$m_{mod(i)} = A_i \left[\rho_g \int_0^L \alpha \cdot dl + \rho_f \int_0^L (1 - \alpha) \cdot dl \right] \qquad [4.9]$$

The void fraction is generally a function of quality so it is desired to transform the integral over the length of the module to an integral over the quality of the module. In order to do this, an assumption regarding the heat flow must be made to determine the relationship between the quality and the tube length [9]. The mass integral equation [4.9] is then normalized with this function and integrated over the quality. The total heat transferred from a module can be represented by the following equation,

$$Q = \int_{x_i}^{x_o} \dot{m}_{R\,mod}\, h_{fg}\, dx = \int_0^L f_Q(x)\, dl \qquad [4.10]$$

Recognizing that the refrigerant mass flow rate and the latent heat of vaporization are constants, the integral in equation [4.7] for the mass of the vapor can be normalized and the result is a heat flux averaged void fraction W_g.

$$W_g = \frac{\int_{x_i}^{x_o} \left[\alpha(x) / f_Q(x) \right] dx}{\int_{x_i}^{x_o} \left[1 / f_Q(x) \right] dx} \qquad [4.11]$$

Equation [4.9] then becomes,

$$m_{mod(i)} = A_i\, L_{mod(i)} \left[\rho_g\, W_g + \rho_f \left(1 - W_g \right) \right] \qquad [4.12]$$

Now the calculation of the module mass is dependent on evaluating the integral in equation [4.11]. In order to do this, as stated earlier, an assumption regarding the heat flux along the length of the tube must be made. The simplest assumption which can be made is that the heat flux is constant, in

which case, it drops out of the integral leaving,

$$W_{g(i)} = \frac{\int_{x_{in(i)}}^{x_{out(i)}} \alpha \, dx}{x_{out(i)} - x_{in(i)}} \qquad [4.13]$$

This assumption is dependent on their being a linear relationship between the quality and tube length. The design of the simulation program is well suited for this assumption if a large number of modules is used because the change in quality along the length of the module is small and can be approximated by a linear distribution without much error. So the constant heat flux assumption is used here which leaves only the void fraction, alpha, to be defined in the integral equation. The next section outlines the void fraction correlations used for this comparison.

4.2 Void Fraction Correlations

The void fraction correlations available in the literature were reviewed by Rice [9] who categorized them into four groups:

1. Homogeneous
2. Slip ratio correlated
3. X_{tt} correlated
4. Mass flux dependent

The Domanski-Didion correlation can be categorized as X_{tt} correlated while the Premoli and Hughmark correlations are categorized as mass flux dependent. The Tandon correlation is both X_{tt} correlated and mass flux dependent. No slip ratio correlated void fraction models are used in the simulation program.

4.2a Homogeneous

The homogeneous void fraction is the most simplistic form available. It is a function of the quality and the vapor-liquid density ratio. The general

form of the equation is given by [21],

$$\alpha = \frac{1}{1 + \left[\frac{1-x}{x}\right]\frac{\rho_g}{\rho_f}} \qquad [4.14]$$

This void fraction model, along with the constant heat flux assumption, can be integrated over the quality change of the module to obtain a closed form solution which is used to verify the numerical integration solutions. This solution is provided in Section 4.3.

4.2b Domanski and Didion

The Domanski and Didion [11] void fraction correlation is based on the Lockhart-Martinelli correlating parameter X_{tt} and the data available from the pressure drop work of Lockhart-Martinelli [22]. It was developed by Domanski and Didion for use in a heat pump simulation. The general formulation is,

$$\alpha = \left[1 + X_{tt}^{0.8}\right]^{-0.378} \qquad \text{for } X_{tt} \leq 10 \qquad [4.15]$$

and,

$$\alpha = 0.823 - 0.157 \ln X_{tt} \qquad \text{for } X_{tt} > 10 \qquad [4.16]$$

where,

$$X_{tt} = \left[\frac{1-x}{x}\right]^{0.9}\left[\frac{\rho_g}{\rho_f}\right]^{0.5}\left[\frac{\mu_f}{\mu_g}\right]^{0.1} \qquad [4.17]$$

4.2c Tandon

The Tandon et. al. [10] void fraction equation includes both the effects of frictional pressure drop and mass flux by correlating as a function of the liquid Reynolds number and the Lockhart-Martinelli correlating parameter.

The void fraction equations are,

$$\alpha = \left[1 - \frac{1.928\, Re_f^{-0.315}}{F(X_{tt})} + \frac{0.9293\, Re_f^{-0.63}}{F(X_{tt})^2}\right] \qquad \text{for } 50 < Re_f < 1125 \quad [4.18]$$

or,

$$\alpha = \left[1 - \frac{0.38\, Re_f^{-0.088}}{F(X_{tt})} + \frac{0.0361\, Re_f^{-0.176}}{F(X_{tt})^2}\right] \qquad \text{for } Re_f \geq 1125 \quad [4.19]$$

where,

$$F(X_{tt}) = 0.15\left[1 / X_{tt} + 2.85 / X_{tt}^{0.476}\right] \quad [4.20]$$

and

$$Re_f = \frac{G_{ref} \times D_i}{\mu_f} \quad [4.21]$$

4.2d Premoli

The Premoli et al. [12] correlation is a function of the liquid Reynolds number, Weber number and the surface tension. It is given by,

$$\alpha = \frac{1}{1 + \left[\frac{1-x}{x}\right] \times \left[\frac{\rho_g}{\rho_f}\right] \times \left[1 + F_1\left(\frac{y}{1+yF_2} - yF_2\right)^{0.5}\right]} \quad [4.22]$$

where,

$$F_1 = 1.578\, Re_f^{-0.19}\left[\rho_f/\rho_g\right]^{0.22} \quad [4.23]$$

$$F_2 = 0.0273\, We_f\, Re_f^{-0.51}\left(\rho_f/\rho_g\right)^{-0.08} \quad [4.24]$$

$$Re_f = \frac{G_{ref} \times D_i}{\mu_f} \quad [4.25]$$

$$We_f = \frac{G_{ref}^2 \times D_i}{\sigma\, \rho_f g_c} \quad [4.26]$$

$$y = \frac{\beta}{1-\beta} \quad [4.27]$$

$$\beta = \frac{1}{1 + \left[\frac{1-x}{x}\right]\frac{\rho_g}{\rho_f}} \quad [4.28]$$

Here we start to see that the integrals to evaluate W_g can get quite complicated as the void fraction models become more complex. It is not clear if a closed form solution even exists for these integral equations, therefore, numerical methods of evaluating the integrals are discussed in Section 4.3.

4.2e Hughmark

The Hughmark void fraction correlation is one the most complex and requires an iterative solution. The correlation is given by,

$$\alpha = \frac{K_H}{1 + \left[\frac{1-x}{x}\right]\frac{\rho_g}{\rho_f}} \qquad [4.29]$$

where K_H is a function of the correlating parameter Z. These values are tabulated in Table 4.1. The correlating parameter Z is given by,

$$Z = \frac{Re_\alpha^{1/6}\, Fr^{1.8}}{y_L^{1.4}} \qquad [4.30]$$

where,

$$Re_\alpha = \frac{D_i\, G_{ref}}{\mu_f + \alpha(\mu_g - \mu_f)} \qquad [4.31]$$

$$Fr = \frac{1}{g_c\, D_i}\left[\frac{G_{ref}\, x}{\beta\, \rho_g}\right] \qquad [4.32]$$

$$y_L = 1 - \beta \qquad [4.33]$$

$$\beta = \frac{1}{1 + \left[\frac{1-x}{x}\right]\frac{\rho_g}{\rho_g}} \qquad [4.34]$$

Table 4.1
Hughmark Flow Parameter Data

Z	K_H
1.3	0.185
1.5	0.225
2.0	0.325
3.0	0.490
4.0	0.605
5.0	0.675
6.0	0.720
8.0	0.767
10.0	0.780
15.0	0.808
20.0	0.830
40.0	0.880
70.0	0.930
130.0	0.980

These are the five void fraction correlations which are added to the condenser simulation program and are used in the comparison of the

predicted condenser charge. The next section reviews the solution techniques used to evaluate the integral W_g.

4.3 Integral Solution Techniques

The calculation of the refrigerant mass in each module of the condenser is dependent on evaluating the integral equation for W_g for each void fraction model. Three solution techniques were explored, a closed form expression, a numerical average of the inlet and outlet void fraction values and a numerical integration using Simpson's rule. A closed form expression for the homogeneous void fraction correlation was found and is useful in verifying the results obtained for the numerical solution techniques. For the other void fraction correlations, only the numerical solution methods are used. This section describes the procedure and assumptions used for these solution techniques.

4.3a Closed Form

The closed form expression is the exact solution for the heat flux averaged void fraction for the module. As the void fraction correlations increase in complexity, however, this expression is more difficult to find if it exists at all. For the homogeneous model, the integration was performed to obtain the following expression,

$$W_g = \left| \frac{x}{c_2} - \frac{c_1}{c_2^2} ln(c_1 + c_2 x) \right|_{x_{in}}^{x_{out}} \qquad [4.35]$$

where,

$$c_1 = \frac{\rho_g}{\rho_f} \qquad \text{and,} \qquad c_2 = 1 - c_1 \qquad [4.36]$$

This closed form solution is used to determine the accuracy of the numerical techniques.

4.3b Numerical Average

The numerical average technique consists of taking a numerical average of alpha at the inlet and outlet of the module. The assumption made here is that there is a linear quality distribution along the module length. This is the same assumption which was made for the use of the constant heat flux assumption in the formulation of W_g in Section 4.1 and similar reasoning applies here. For a small module length, relative to the total length of the condenser, the change in quality from the inlet to the outlet is small and W_g can be approximated by an average of the inlet and outlet void fraction. Clearly, the accuracy of this technique is dependent on this assumption being satisfied. If the entire two phase region of the condenser is being modeled with one module, it is more appropriate to use a numerical integration method.

4.3c Simpson's Rule

Simpson's rule is a numerical integration method used to approximate solutions to integral equations. With this method, the integral interval is divided into equally spaced points at which the function is evaluated. The integral is then approximated by combining the function evaluations based on a quadratic polynomial method. The general mathematical formulation for Simpson's rule is given by,

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right] \qquad [4.37]$$

This method is compared with both the closed form solution and the numerical average in the results chapter.

5.0 PARAMETER ESTIMATION

As stated in the introduction, the prediction of refrigerant charge in the condenser coil was heavily influenced by the prediction of the subcooling length of the condenser which will be verified by the results presented in Chapter 6. It was also observed, that even with acceptable accuracy in the prediction of the overall coil heat capacity, a large error in the subcooling prediction occurred. In an attempt to minimize this error, some parameters in the model were adjusted using optimization methods. The purpose of this chapter is to provide a detailed account of these methods and the parameter estimation results. This chapter begins with the selection of the search parameters which were varied in the simulation program. The two optimization methods used to vary the parameters are outlined in Section 5.2. The next section provides a discussion of the objective functions chosen for the subcooling minimization and the last section contains a discussion of the search results along with the selection of the parameter set used for the inventory analysis in Chapter 6.

5.1 Search Parameters

The first step in the optimization process was the selection of the search parameters which were adjusted in the condenser simulation program. As stated previously, the overall objective of this process was to minimize the error in the subcooling prediction and it was reasoned that this could be accomplished by properly predicting the condenser outlet conditions (temperature, pressure and enthalpy). These outlet conditions are heavily dependent on the heat transfer and pressure drop correlations used in the simulation, therefore, the parameters were selected to replace the constant variables in these correlations.

The heat transfer correlations which were modified for the optimization are the Colburn j-factor correlation for the air side heat transfer coefficient and the Dobson correlation for the refrigerant side two phase heat transfer coefficient. Each of these correlations contain two constants which were replaced with search parameters. For the pressure drop determination, the Souza et.al. correlation was modified to contain two parameters for the search process. This provided a total of six parameters which were selected for optimization, four heat transfer parameters and two pressure drop parameters.

In the j-factor air side heat transfer coefficient correlation, equation [2.16], the two constants found using the modified Wilson plot technique were set as parameters P1 and P2. This is given by,

$$j = \mathrm{P1}\ Re_{air}^{-\mathrm{P2}} \qquad [5.01]$$

For the Dobson correlation, the two constants were set as parameters P3 and P4. This equation is given by,

$$h_{ref(i)} = \frac{\mathrm{P3} \times h_{f(i)}}{X_{tt(i)}^{\mathrm{P4}}} \qquad [5.02]$$

For the pressure drop parameters, the two constants C1 and C2 in equation [2.54] were replaced with search parameters five and six (P5 and P6).

Only the refrigerant two phase heat transfer coefficient and pressure drop correlations were selected to have search parameters. This was due to the fact that the majority of the condenser coil was in the condensing region, which implied that the error incorporated in the single phase regions do not contribute as much to the overall error in the predicted outlet conditions. On the other hand, the addition of four more search parameters would drastically increase the complexity of the optimization. The next section outlines the optimization methods which were implemented into the simulation program to perform the subcooling minimization.

5.2 Optimization Methods

Two optimization search methods were used with the condenser simulation program to minimize the predicted amount of subcooling at the condenser outlet. They are a Marquardt search and an exhaustive search. This section provides a description of these two techniques.

5.2a Marquardt Search

The Marquardt optimization search used in the simulation program was written by VanderZee [28] for parameter estimation in an empirical condenser model. This Marquardt search program is generalized enough that it was easily implemented into the condenser simulation program. The general mathematical formulation of the Marquardt optimization search is as follows. The search begins with an initial guess for each of the parameters $\mathbf{X}_k$ and a defined objective function for minimization $f(\mathbf{X}_k)$. For each iteration in the search, a direction is determined by using the Hessian. The general mathematical formulation for the search direction matrix is given by,

$$\mathbf{S}_k = -\left[\nabla^2 f(\mathbf{X}_k) + \lambda_k \mathbf{I}\right]^{-1} \nabla f(\mathbf{X}_k) \qquad [5.03]$$

where,

$\mathbf{S}_k$ = Search direction matrix
$\nabla^2 f(\mathbf{X}_k)$ = Numerical second derivative matrix (Hessian)
λ_k = Scaling factor
$\mathbf{I}$ = Identity Matrix
$\nabla f(\mathbf{X}_k)$ = Numerical first derivative matrix

Once the direction is determined, the search takes as many steps as needed in that direction until the minimum is found. The search step matrix is given by,

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \Delta\mathbf{X}_k = \mathbf{X}_k + \alpha_k \mathbf{S}_k \qquad [5.04]$$

where,

α_k = Scaling factor

Once the minimum in the calculated direction is determined, a new direction is calculated and the search continues until a convergence criterion is satisfied.

The mathematical formulation of the Marquardt search provides for some potential problems in it's application to real problems. In equations [5.03] and [5.04], the scaling factors (α_k and λ_k) are used to weight the functions and can be set to vary as the search progresses or to remain constant. The selection of these scaling factors is influential to the successful operation of this search technique and introduces some potential complications in it's control. One additional source of potential problems with the Marquardt search, is that the mathematical formulation is based on the assumption of a continuous objective function. Due the module structure of the program, it is not clear that the objective function will indeed be continuous. This is discussed in greater detail in the search results section.

The number of parameters used in the Marquardt search also has an impact on the convergence results. This is due, in part, to the fact that the same scaling factor is applied to the step size for each parameter. To avoid such complications, measures were taken to reduce the number of search parameters. This is discussed in greater detail in the results section. Due to some of the problems encountered with the Marquardt search technique, an exhaustive search routine was also implemented into the condenser simulation program. The main purpose of this technique was to verify the Marquardt search results. This search technique is described in the next section.

5.2b Exhaustive Search

The exhaustive search technique was utilized to verify the results obtained from the Marquardt search and as a stand alone search technique. Although it does not provide the speed of the Marquardt search, it also does not have the intricacies which constrain its use. The procedure used for the exhaustive search is as follows. An initial parameter guess and interval step size are specified by the user for each parameter which allows for greater control over the step size than is possible with the Marquardt search. The user also specifies the number of steps to be taken at each iteration. Starting with the first parameter, the search takes the specified number of steps in the positive and negative directions. The minimum for the parameter is determined and that parameter is updated before searching on the next parameter. This continues until all parameters are searched on which constitutes one iteration. As the search continues, the search step sizes are decreased when no change in any of the parameters takes place for one iteration. This continues until the convergence criterion is met or until the maximum number of iterations are performed.

Overall, it is a fairly crude search technique, however, it does allow the behavior of the objective function to be analyzed. Additionally, it should be noted that although it is referred to as an exhaustive search, it is not a complete exhaustive search which would require the objective function to be evaluated for all possible combinations of the parameters. The use of a complete exhaustive search technique was ruled out due to the lengthy run time of the simulation program. The next section provides a discussion of the various objective functions which were selected for use in the minimization searches.

5.3 Objective Functions

One of the most essential elements of the parameter estimation process was the selection of an appropriate objective function. As state previously, the intention was to minimize the error in the predicted amount of subcooling which is a result of the error in the predicted condenser outlet conditions (temperature, pressure and enthalpy). With this in mind, three error functions were selected to be minimized; the total heat transfer, the total pressure drop and the degrees of subcooling at the condenser exit. The degrees of subcooling at the condenser exit was selected to use as an error function instead of the length of the subcooling region because experimental data for this value was available.

The main format chosen for the objective function is the sum of the least squares error given by,

$$\text{SUM} = \sum_{1}^{Ndata} \left(\frac{M_{exp} - M_{sim}}{M_{exp}} \right)^2 \qquad [5.05]$$

$$\text{OBJ} = \left[\frac{\text{SUM}}{N_{data}} \right]^{0.5} \qquad [5.06]$$

where,

OBJ = Objective Function
M_{exp} = Experimental Value
M_{sim} = Simulation Value
N_{data} = Number of Data Points

Replacing the chosen values for the error functions into equation [5.05] gives,

$$\text{SUM}_Q = \sum_{1}^{Ndata} \left(\frac{Q_{exp} - Q_{sim}}{Q_{exp}} \right)^2 \qquad [5.07]$$

$$\text{SUM}_{\Delta P} = \sum_{1}^{Ndata} \left(\frac{\Delta P_{exp} - \Delta P_{sim}}{\Delta P_{exp}} \right)^2 \qquad [5.08]$$

$$SUM_{\Delta Tsub} = \sum_{1}^{Ndata} \left(\frac{\Delta Tsub_{exp} - \Delta Tsub_{sim}}{\Delta Tsub_{exp}} \right)^2 \qquad [5.09]$$

Equations [5.07] to [5.09] are referred to as error functions throughout this thesis.

The objective function in the minimization search was expressed as either one of the error functions in equations [5.07] to [5.09] or as a combination of the error functions. As is shown in the search results section, using one of the error functions for the objective function did not always achieve the desired outcome. For example, if the objective function was chosen as the total pressure drop only, the search program minimizes this variable without regard for either the heat transfer or degrees of subcooling. In this process, a greater error in both the heat transfer and subcooling can occur.

When combining the error functions for the objective function another problem arises which is the relative magnitude of the error functions. For example, when running the simulation program with the j-factor correlation for the air side heat transfer coefficient, the Dobson correlation for the refrigerant side heat transfer coefficient and the Lockhart-Martinelli correlation for the two phase pressure drop, the simulation predicts an average error in Q of 5.22%, an average error in ΔP of 30.41% and an average of 10.32°F subcooling (the experimental data analysis calculates an average of 1.1°F subcooling at the condenser outlet). It is obvious that when combining these error functions an appropriate weighting factor must be applied so that equal consideration is given to each error function. This becomes more difficult as the search progresses and the relative magnitudes of the error functions change.

The weighting of each component of the objective function is also dependent on the accuracy of the experimental measurements which are used to determine the error in the simulation. For example, there is higher uncertainty in the pressure drop measurement, therefore, it may be decided that this term should not be weighted as heavily as the error functions for the heat capacity and degrees of subcooling. To reduce some of the variance in the error functions, a data subset was selected for the search analysis. This had the added advantage of drastically reducing the run time of the search program. Experimental data was gathered for forty-six different condenser inlet conditions and the data subset selected contained twenty of these points.

The first points that were eliminated from the original data set were points for which the simulation predicted a very large error in the total pressure drop across the coil (ΔP_{tot}). These points occurred at low refrigerant mass flow rates and the error in the ΔP_{tot} prediction was over 45% using the Dobson correlation. The next criteria used for the elimination of experimental data points was the total capacity of the coil Q. Points on both the high and low ranges of Q were eliminated. Additional data points were eliminated at random to reduce the experimental data subset to twenty points. This subset is provided in Appendix D.

5.4 Search Results

In the previous sections of this chapter, the minimization search techniques, search parameters and objective functions have been discussed. This section provides a discussion of the results for the two search techniques used to minimize the predicted subcooling in the condenser. The limitations of each search technique are explored and a set of parameters for the inventory analysis is selected.

5.4a Marquardt Search

The Marquardt search routine written by VanderZee [28] was implemented into the Module Based Condenser Simulation program. The first element of running the Marquardt search was the selection of the scaling factors λ_k and α_k. As stated previously, the success of the Marquardt search is dependent on the proper selection of these scaling factors which is complicated by the fact that no information regarding the behavior of the objective functions for this problem was available. The selection process was, for the most part, one of trial and error.

One typical function used for λ_k, is to start with a large number (say 1.0E05) and to decrease this by an order of magnitude with each iteration. This function puts more weight on the first derivatives of the objective function initially, and gradually shifts the weight to the second derivative values as the number of iterations increase. The initial value of λ_k must still be properly selected.

The selection of an appropriate function for the scaling factor α_k is more obscure than that for λ_k. For λ_k, the scaling factor has the mathematical function of weighting the first and second derivatives, however, for α_k the function is weighting the search step size for all parameters. Again, the general reasoning is to begin with a large value for α_k and to decrease that value as the number of iterations increase. This has the effect of taking large step sizes initially and then decreasing the step size as the search progresses. Additionally, the number of parameters being searched on can effect the scaling factor α_k. As the number of parameters increase, it becomes more difficult to find a function which will properly scale all the step sizes. Due to the lack of knowledge regarding the behavior of the objective function, it is difficult to properly select a scaling function for α_k. VanderZee found it

useful to set this scaling factor as a function of the change in the objective function from iteration to iteration so that as the change in the objective function decreased, the scaling factor decreased. This type of function was not well suited for this problem (an observation made through trial and error). Overall, it was found that leaving the scaling function α_k as a constant provided better results.

The next determination made before the Marquardt search was run was the objective function to use. As has been stated previously, the main objective of the search process was to minimize the error in the predicted subcooling which is dependent on properly predicting the condenser outlet conditions (temperature, pressure and enthalpy). This suggests that the overall objective may be achieved by using only the heat transfer and pressure drop error functions, therefore, the search process began with the use of these error functions.

The first Marquardt search consisted of a search on all six parameters with the objective function set to combination of the heat transfer error term and the total pressure drop error term given by,

$$OBJ = 0.9\left[\frac{SUM_Q}{N_{data}}\right]^{0.5} + 0.1\left[\frac{SUM_{\Delta P}}{N_{data}}\right]^{0.5} \qquad [5.10]$$

The selection of this objective function weighted the heat transfer error term slightly more than the pressure drop error term. This was done to reduce the influence of the experimental error in the pressure drop measurement by not weighting that term as heavily as the heat capacity term. The results for this search showed minimal improvement in the objective function. Various scaling factors were tried to improve the Marquardt search results, however, the overall results were not drastically improved. It is interesting to note, however, that the Marquardt search converged on different values for the

objective function and search parameters when the scaling functions were changed. This indicated the sensitivity to the scaling functions.

The trial and error process undergone for the first search produced two important issues which needed to be addressed. The first issue was whether or not the Marquardt search technique was indeed converging on the minimum value of the objective function and at that point there was no way of making this determination. The only evidence which indicated that it was not finding the minimum, was that for different scaling functions the Marquardt search converged on different parameter values. This led to the second issue of how much improvement in the heat transfer and pressure drop error functions was reasonable to expect. The results obtained by using the j-factor air side correlation and the Dobson refrigerant side correlation led to an average error in Q of 5.22%, an average error in ΔP_{tot} of 30.41% and an average of 10.32°F subcooling. The error in ΔP_{tot} is rather high, but the error in Q is certainly within acceptable limits for an empirical correlation, let alone a generalized model. So at this point it was unclear as to the amount of improvement which could be expected.

The next search consisted of a search on all six parameters using only one data point. The objective function used was the same and is given by equation [5.10]. The purpose of this search was to determine how well the Marquardt search was working. By searching on only one data point, the program was very fast and could be analyzed quickly. It was found for this search that the value of the objective function moved from 0.06913 to 0.0336. This was reasonable, however, for only one data point it was believed that the parameters should be able to predict the heat transfer and pressure drop very accurately. As stated earlier, another factor influencing the Marquardt search is the number of parameters, therefore, the next search attempt was to reduce

the number of parameters.

To reduce the number of search parameters the following strategy was employed. First, a search was conducted on the two pressure drop parameters using only the pressure drop error term as the objective function,

$$OBJ = \left[\frac{SUM_{\Delta P}}{N_{data}} \right]^{0.5} \qquad [5.11]$$

The pressure drop parameters (P5 and P6) which minimized this objective function were then used as constants in a search on the four heat transfer parameters using only the heat capacity error term as the objective function,

$$OBJ = \left[\frac{SUM_{Q}}{N_{data}} \right]^{0.5} \qquad [5.12]$$

The two searches were alternated until a minimum had been reached. For the one data point search, this method worked extremely well. The objective function for the pressure drop parameters went from 0.29055 to 0.0000 and for the heat transfer parameters went from 0.05600 to 6.955E-08. One additional piece of information provided by this search was that by eliminating the error in the heat transfer and pressure drop predictions, the error in the predicted amount of subcooling was eliminated. This was very encouraging, however, as will be shown, the search results obtained using the entire data subset were not as accurate.

The next search applied this same strategy to the entire data subset. The best set of parameters obtained with the Marquardt search are provided in Table 5.1. The predicted values by the simulation program with these parameters provided the following results. There was an average error in Q of 4.60%, an average error in ΔP_{tot} of 24.47% and an average of 8.20°F subcooling at the condenser outlet (experimental values determined an average of 1.11°F subcooling at the condenser outlet). These parameters show

an improvement in the prediction of the heat capacity, pressure drop and degrees of subcooling, however, there is still a substantial amount of subcooling predicted.

Marquardt Search Parameters			
Parameter	Value	Parameter	Value
1	0.2105268	4	1.1568509
2	0.4414812	5	14.8649976
3	2.4015876	6	1.6250237

Table 5.1 - Marquardt Search Parameters

Up until this point, only the heat capacity and pressure drop error terms have been examined in the objective function and although they provided excellent results for the search on one data point, the results obtained for the entire data subset did not eliminate the error in the subcooling prediction. This led to the use of the degrees of subcooling objective function, however, it could not be used in the Marquardt search, due to discontinuities in the objective function. The exhaustive search was used to minimize the degrees of subcooling objective function and the results are provided in the next section. Additionally, the exhaustive search was used to verify the Marquardt search results for the heat transfer and pressure drop objective functions.

5.4b Exhaustive Search

The exhaustive search was developed for the following reasons; to verify the Marquardt search results, to handle the discontinuities in the degrees of subcooling objective function and for use with a combined objective function. To verify the results obtained from the Marquardt search

for the pressure drop and heat capacity error functions the exhaustive search was run using the same strategy as the Marquardt search. First the pressure drop parameters were searched on and then the heat transfer parameters were searched on. The exhaustive search for these error functions did not produce a better set of parameters than the Marquardt search.

To use the degrees of subcooling as the objective function, the exhaustive search technique was necessary. As stated previously, there are discontinuities in this objective function which prohibit the use of the Marquardt search. For example, if the amount of subcooling at the outlet of the condenser is thought of as only a function of the total heat transfer, a graph similar to Figure 5.1 can be produced.

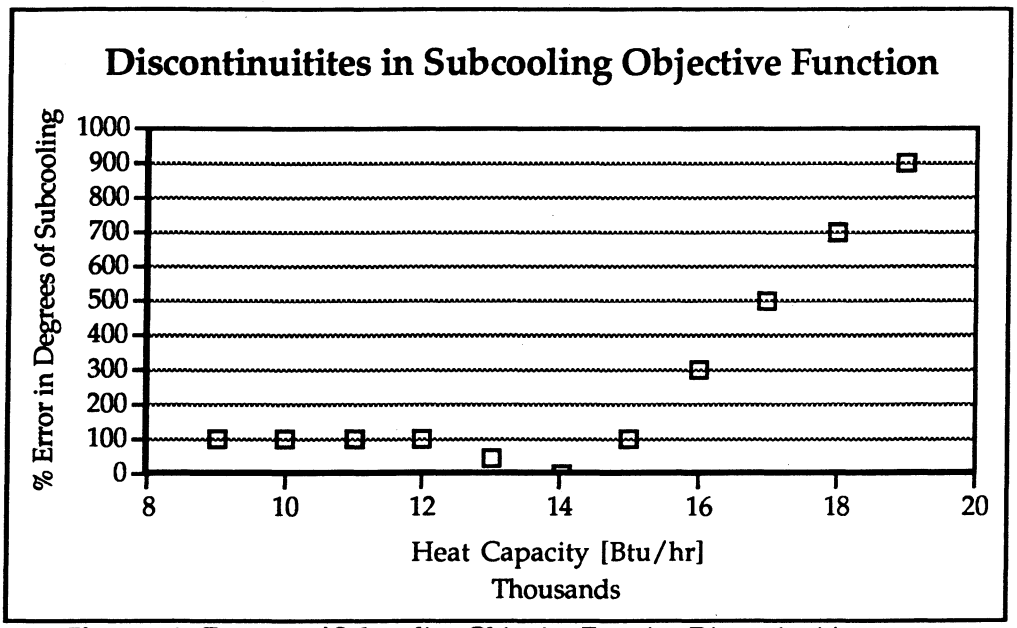

Figure 5.1 - Degrees of Subcooling Objective Function Discontinuities

This graph is an illustration of the behavior of the objective function and not a result of actual data. When the predicted heat capacity of the simulation is less than that of an actual data point (assuming the pressure drop is correct) there is no subcooling predicted so the error is 100%. This is the case for any heat capacity which produces no subcooling regardless of the exit quality or

the error in the heat capacity. As the heat capacity increases, the degrees of subcooling objective function decreases to zero and then increases again for as long as the heat capacity is increased. In minimizing the degrees of subcooling objective function, once the predicted heat capacity is small enough that no subcooling is predicted by the simulation, the Marquardt search is unable to determine a direction for the parameters because all the numerical derivatives at this point are zero.

This example was contrived to illustrate the problems encountered with the Marquardt search if there are discontinuities in the chosen objective function. To avoid any problems of this nature, the exhaustive search technique described earlier was used to minimize the degrees of subcooling objective function. The parameters obtained from this search are provided in Table 5.2. This set of parameters produced an average error in the heat capacity prediction of 11.50%, in the total pressure drop of 27.38% and an average of 0.73°F subcooling at the condenser outlet. This shows that the subcooling was minimized, however in the process, the error in the heat capacity prediction drastically increased and the pressure drop prediction increased slightly. This suggests that a combination of the degrees of subcooling and heat capacity error functions is necessary to obtain the desired results.

Degrees of Subcooling Parameters			
Parameter	Value	Parameter	Value
1	0.1156	4	0.8000
2	0.4038	5	7.2884
3	2.7100	6	1.6550

Table 5.2 - Exhaustive Search Parameters - Degrees of Subcooling Objective Function

The combined objective function selected for the exhaustive search is given by,

$$OBJ = \left[\frac{SUM_Q}{N_{data}}\right]^{0.5} + 0.01\left[\frac{SUM_{\Delta Tsub}}{N_{data}}\right]^{0.5} \quad [5.13]$$

This objective function puts more weight on the degrees of subcooling error function at first due the large error in the subcooling prediction. Then the weight is shifted to the heat capacity error function as the search continues and the error in the degrees of subcooling decreases. The parameters obtained using this objective function are given in Table 5.3. These parameters provided results with an average error in the heat capacity prediction of 7.54%, the average error in the total pressure drop prediction was 26.82% and the average degrees of subcooling at the condenser outlet were 0.82°F.

Combined Exhaustive Search Parameters			
Parameter	Value	Parameter	Value
1	0.172943	4	2.156851
2	0.441481	5	14.864998
3	2.243828	6	1.625024

Table 5.3 - Exhaustive Search Parameters - Combined Objective Function

The results obtained from this last set of parameters indicate that the best objective function for this problem is the combined heat capacity and degrees of subcooling objective function. These parameters allow the error in the predicted subcooling to be minimized without drastically increasing the error in the heat capacity and pressure drop predictions, as was the case for the parameters found using only the degrees of subcooling as the objective function. Therefore, these parameters were selected for the inventory analysis provided in Chapter 6.0.

6.0 SENSITIVITY RESULTS

This chapter provides a discussion of the simulation results along with a sensitivity analysis of the refrigerant inventory predictions. A comparison of the simulation's predicted heat capacity using the available correlations is given in Section 6.1. The predicted total pressure drop results are provided in Section 6.2 and the charge inventory prediction results for the various void fraction correlations are given in Section 6.3. The goal of the inventory prediction comparison is to establish the factors which most influence the predicted refrigerant charge in the condenser coil.

6.1 Heat Transfer Results

This section presents a discussion of the predicted heat capacity results using the available correlations in the simulation program for the air side and refrigerant side heat transfer coefficients. For the air side, the predicted capacity for the Colburn j-factor and the Temperature Limit Principle correlations (see section 2.3) are compared in Section 6.1a. Then in Section 6.1b, the refrigerant side heat transfer coefficient correlations are compared.

6.1a Air Side Correlation Comparison

There are two air side heat transfer coefficient correlations available in the condenser simulation program. The predicted overall heat capacity by the simulation using these correlations is compared in this section. In general, the Colburn j-factor correlation predicted the heat capacity more accurately than the Temperature Limit Principle correlation for this coil. These results are shown in Figure 6.1 which is a graph of the predicted heat capacity for the air side correlations using the Dobson correlation for the refrigerant side heat transfer coefficient. The average error in the predicted capacity using the j-

factor correlation was 5.22% and was 15.11% for the Temperature Limit Principle correlation.

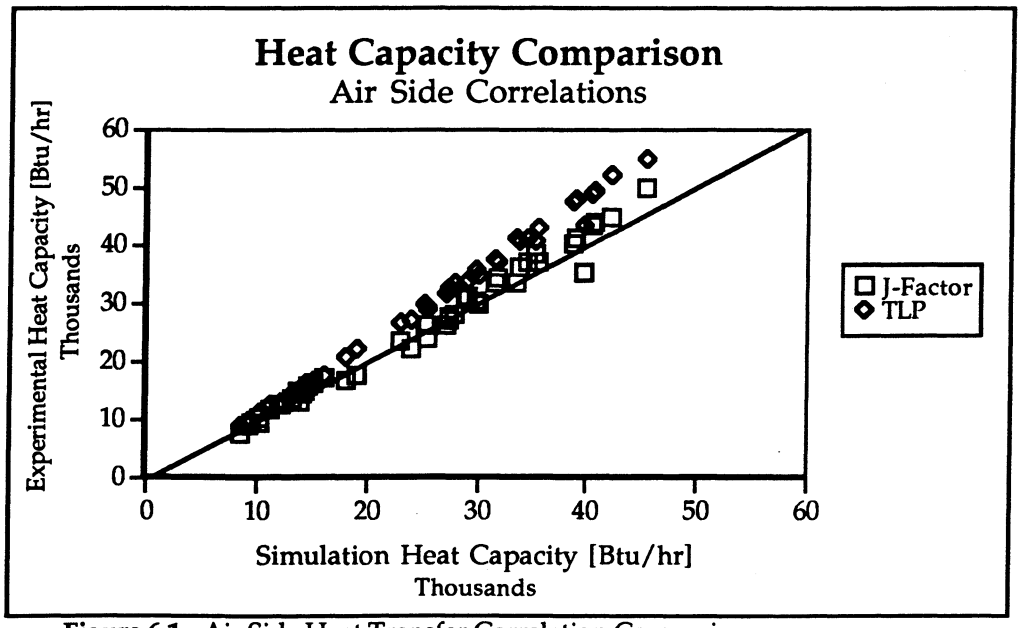

Figure 6.1 - Air Side Heat Transfer Correlation Comparison

This graph also indicates that the error in the heat capacity prediction increased with increasing overall capacity for both correlations. These correlations are empirical and were derived from experimental data sets representing different air inlet properties for constant refrigerant inlet properties. This restriction constrained the experimental data set used to determine the correlation's coefficients to a limited portion of the entire range of heat capacities being analyzed for this condenser coil. The result of this limitation is that the correlations predicted more accurately in the range of heat capacities that were used to determine the coefficients. The coefficients determined by Marin [24] were based on data with experimental heat capacities around 20,000 Btu/hr while the Colburn j-factor coefficients were based on experimental data with heat capacities in the range of 15,000 to 25,000 Btu/hr. This is one possible reason for the greater accuracy of the j-factor correlation.

6.1b Refrigerant Side Correlation Comparison

The following comparison is of the predicted condenser heat capacity using the three refrigerant side heat transfer coefficient correlations (Dobson, Shah and Cavallini). The air side correlation used for this comparison was the Colburn j-factor. The results are shown in Figure 6.2 which compares the predicted heat capacity for the refrigerant two phase correlations. The average error in the heat capacity prediction using the Dobson correlation was 5.22%, it was 6.95% for the Cavallini correlation and 10.60% for the Shah correlation. All of these correlations predicted the heat capacity in an acceptable range of error, however, the Dobson correlation, which was developed specifically for R134a, predicted the overall heat capacity more accurately.

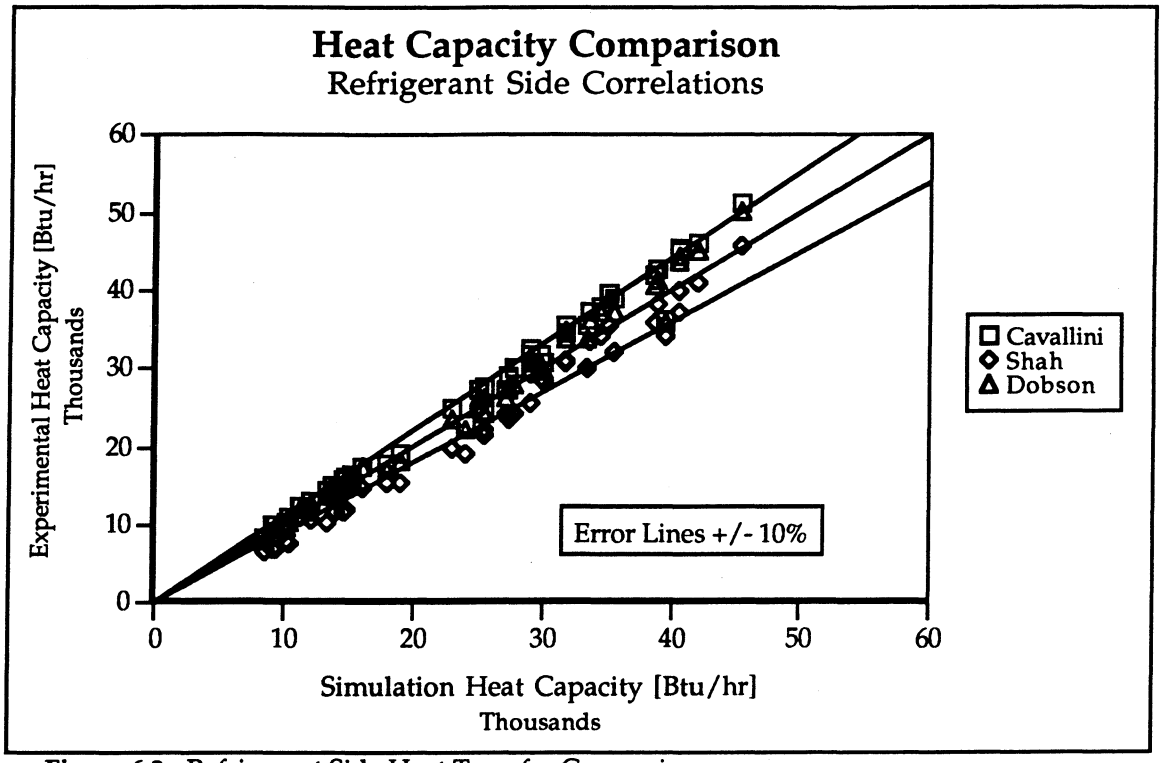

Figure 6.2 - Refrigerant Side Heat Transfer Comparison

In addition, Figure 6.2 indicates that the Cavallini correlation over-predicted the heat capacity while the Shah correlation under-predicted the coil capacity. This accounts for the larger amount of subcooling predicted when using the

Cavallini correlation. The Cavallini correlation predicted an average of 15.5°F subcooling while the Shah correlation predicted an average of only 0.37°F subcooling at the condenser outlet. The Dobson correlation predicted an average of 10.32°F subcooling.

The simulation heat capacity predicted for the search parameters given in Table 5.3 is shown in Figure 6.3. These parameters resulted in an average error of 7.54% for the heat capacity prediction which is slightly higher than the error for the Dobson correlation. Figure 6.3 also indicates that the heat capacity is generally under-predicted which results in the small amount of subcooling which is predicted using these parameters (0.82°F at the condenser outlet).

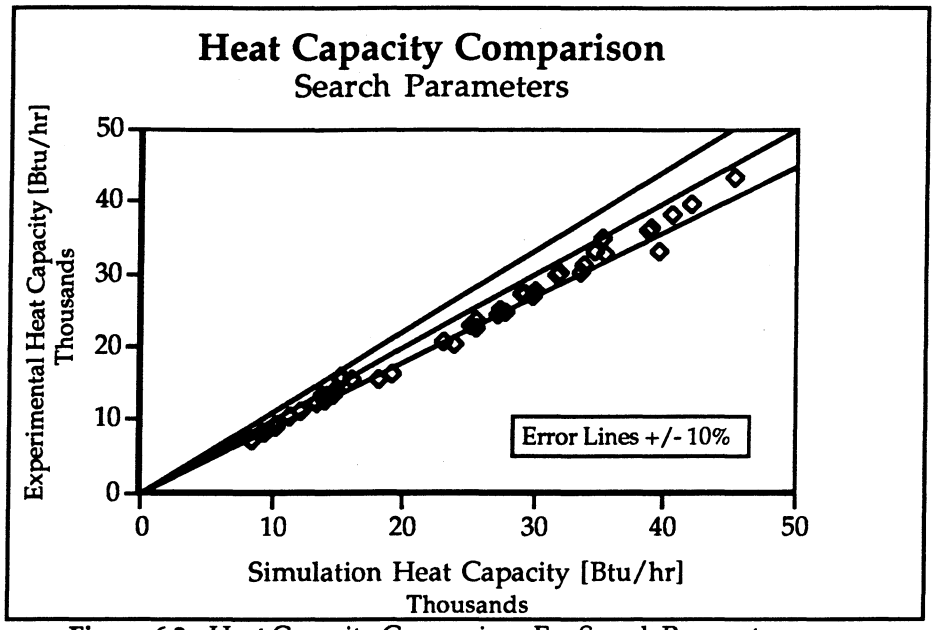

Figure 6.3 - Heat Capacity Comparison For Search Parameters

There are many factors which could account for the error in the simulation's heat capacity prediction. In Section 6.1a, some possible errors in the air side heat transfer correlations were discussed. Additionally, there is a degree of uncertainty in all the experimental measurements which could account for part of the error in the heat capacity prediction. Another possible

source of error could be the refrigerant flow regime. Each of the refrigerant side heat transfer coefficient correlations were developed using data for an annular flow regime and it has not been established whether or not this criterion has been satisfied. In Section 2.6 a method for determining the refrigerant flow regime based on the Weber and Froude numbers was presented. Results for several data points using the criterion outlined in that section are provided below.

The next set of graphs plot the calculated Weber and Froude numbers along the length of the condenser for various air mass flow rates. Using the criteria set by Rahman [27], the flow regime of the refrigerant was determined and labeled on the graph. For this analysis, the Dobson correlation was used for the refrigerant side heat transfer coefficient and the Colburn j-factor correlation was used for the air side heat transfer coefficient. Graphs were plotted for data points four, five and six whose operating conditions are in Appendix B.1. Figure 6.4 shows the flow regimes determined for data point 4.

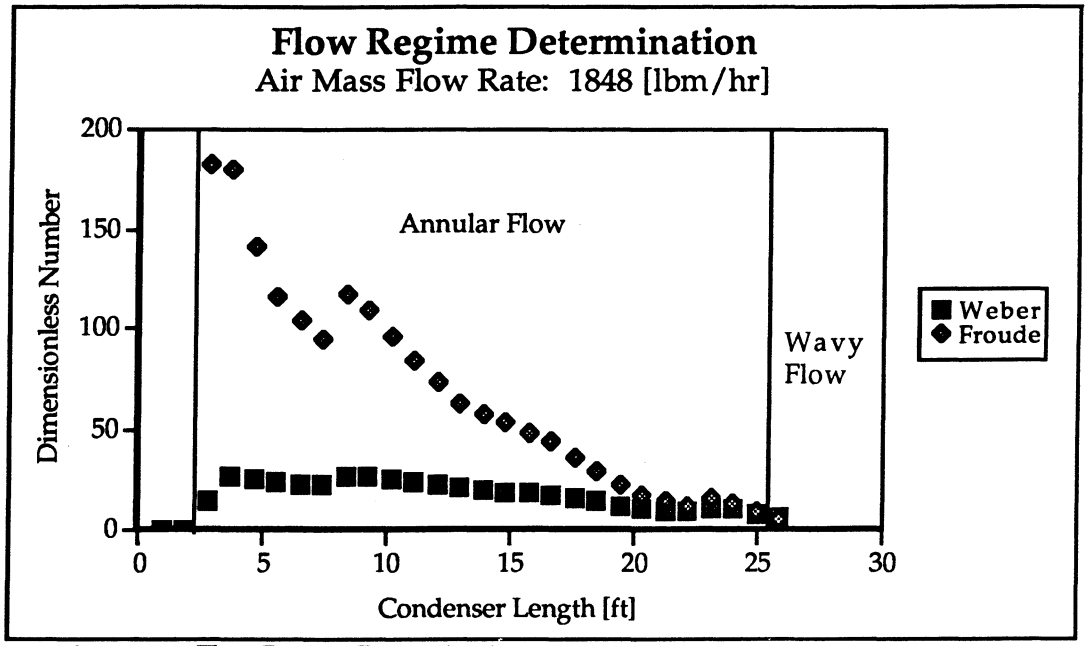

Figure 6.4 - Flow Regime Determination

This figure indicates that the majority of the condenser was considered to be in the annular flow regime and only the last module was in the wavy flow regime. There was no mist flow or subcooling predicted for this data point. Figure 6.5 shows the same comparison for point 5 which has a slightly higher air mass flow rate and a comparable refrigerant mass flow rate. For this data point, most of the refrigerant was in the annular flow regime as was the case for point 4, however, a slightly larger wavy flow regime was determined. There was no mist flow and a very small amount of subcooling predicted for this data point.

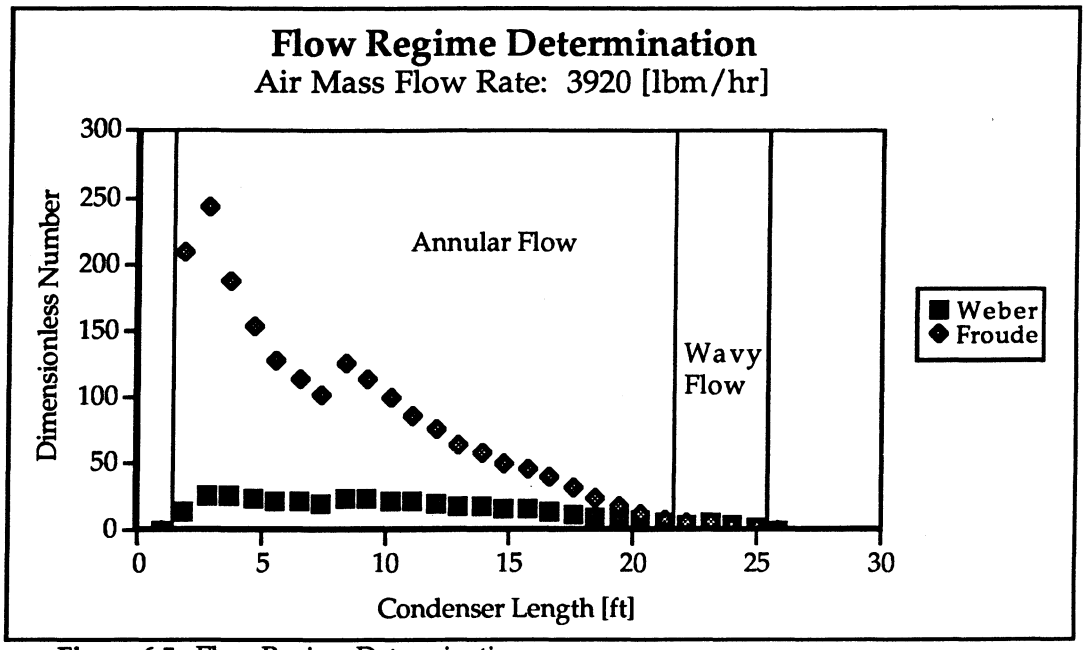

Figure 6.5 - Flow Regime Determination

The last graph, Figure 6.6 shows the flow regimes for point 6 which has a higher air mass flow rate. Similar results were obtained for this data point which shows only a small wavy flow regime and no mist flow regime. Although results are provided for only a small portion of the experimental data points taken, the results for the other data points were similar. Overall, it was found that the majority of the condensing region was determined to be

in the annular flow regime using the criterion set by Rahman. Only a small portion of the condenser was determined to be in the wavy flow regime, and very few data points exhibited any mist flow.

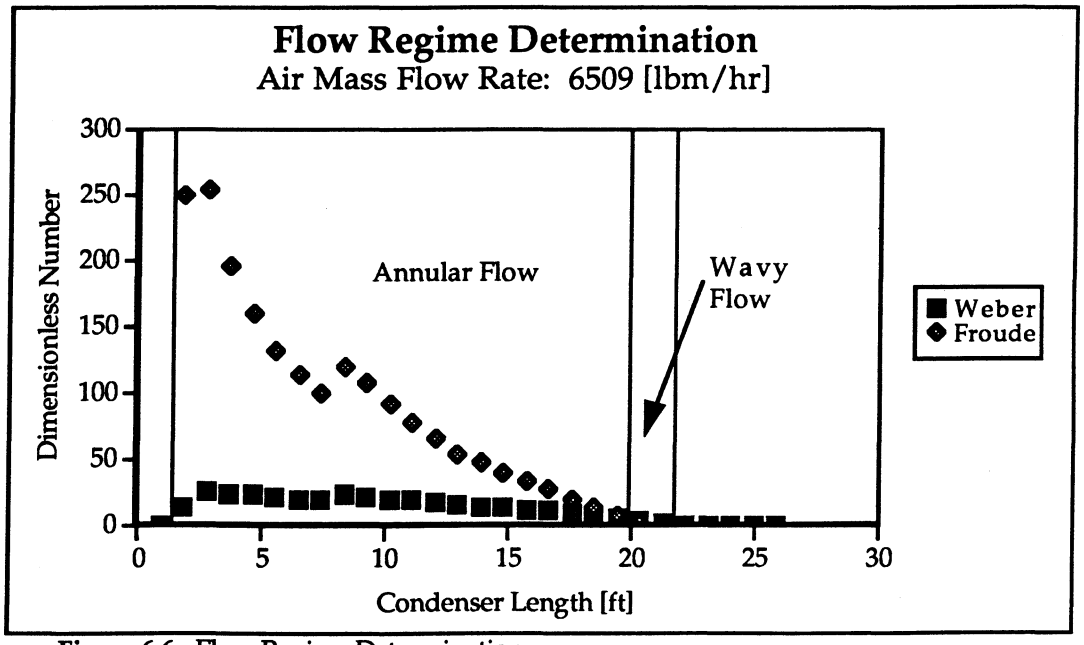

Figure 6.6 - Flow Regime Determination

6.2 Pressure Drop Results

The total pressure drop across the condenser coil is also predicted by the simulation program. The following comparison is of the predicted total pressure drop for the different two phase heat transfer coefficient correlations. The average error in the total pressure drop was determined to be 30.41% for the Dobson correlation, 35.90% for the Cavallini correlation and 23.86% for the Shah correlation. This results are compared in Figure 6.7 which indicates that at small total pressure drops (less than 15 psi), the overall error in the pressure drop prediction is higher than at larger total pressure drops. As was discussed in Chapter 5, the uncertainty in the total pressure drop measurement is magnified at lower pressure drops which was why these points were eliminated from the parameter search data subset.

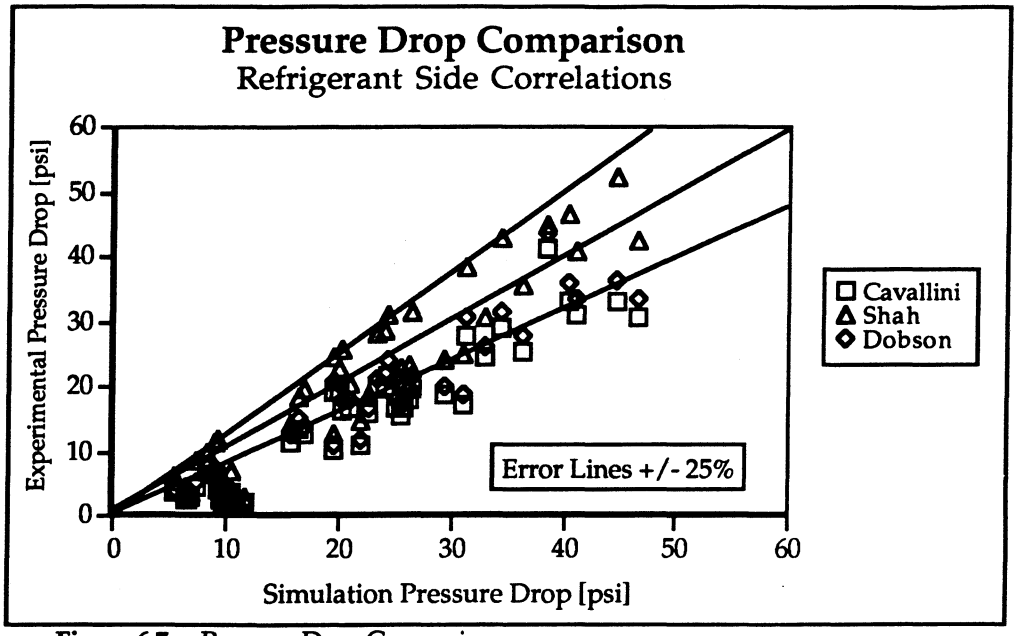

Figure 6.7 - Pressure Drop Comparison

6.3 Inventory Results

This section presents the charge inventory prediction sensitivity analysis. The difference in the predicted values for the various heat transfer correlations are compared, and the results indicate the strong relationship between the amount of subcooling and the predicted refrigerant charge. Other comparisons provided are for the void fraction correlations, the integral solution techniques and the mass distribution along the length of the condenser. The analysis provided is a sensitivity analysis and is not intended to be an indication of which void fraction correlation predicts the refrigerant charge most accurately. Experimental data measuring the condenser mass will be available in the future and will address this issue. The purpose of this work is to establish the factors which most influence the prediction of refrigerant inventory in condenser coils.

6.3a Heat Transfer Correlation Sensitivity

The refrigerant inventory prediction by the condenser simulation program is heavily influenced by the amount of subcooling predicted. This was the instrumental reason for the parameter estimation techniques used in Chapter 5. A comparison of the predicted inventory (for the homogeneous void fraction correlation) using the different two phase heat transfer coefficient correlations is provided in Figure 6.8. This figure clearly illustrates the influence of the predicted subcooling on the inventory prediction.

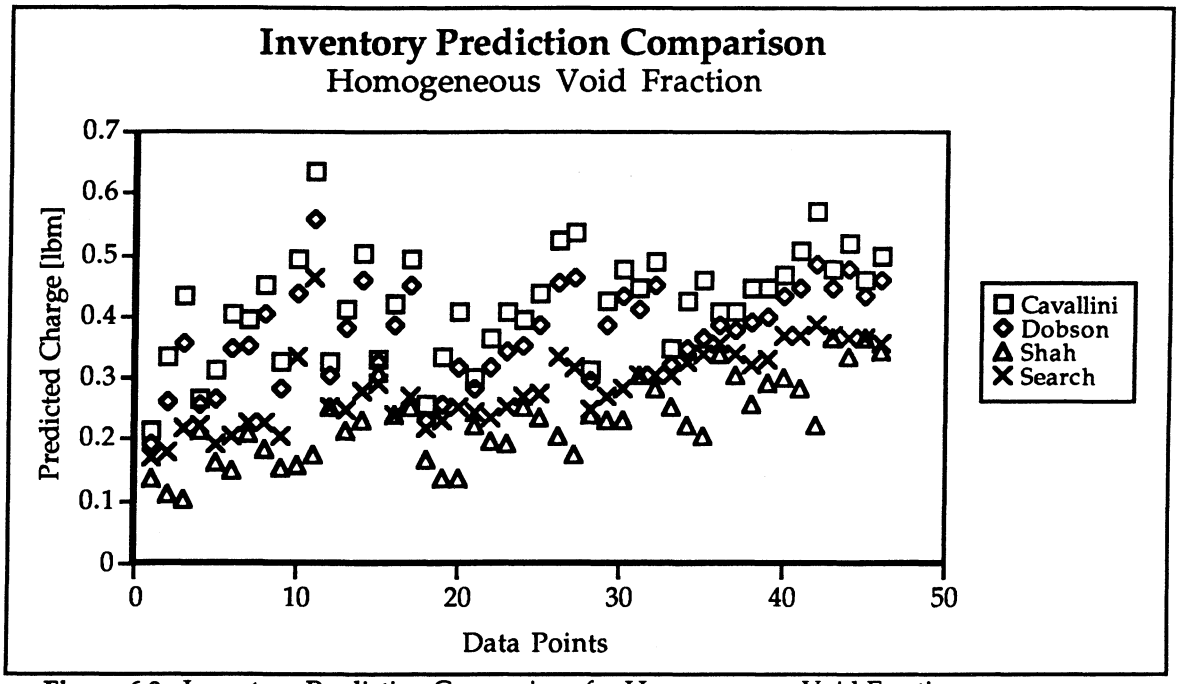

Figure 6.8 - Inventory Prediction Comparison for Homogeneous Void Fraction

The Cavallini correlation which exhibited the largest error in the amount of subcooling (15.5°F) predicted the largest inventory while the Shah correlation which predicted the smallest amount of subcooling (an average 0.37°F) predicted the lowest condenser mass. The Dobson correlation which produced an average of 10.32°F subcooling predicted the next largest inventory and the search parameters which had an average of 0.82°F

subcooling predicted a condenser inventory between the Shah and Dobson predictions. For both the Shah correlation and the search parameters, the amount of subcooling was low, however, there was still a large difference between the inventory prediction for these cases. This is due to the fact that the Shah correlation under predicted the overall coil capacity by a greater amount than the search parameters did. This leads to a higher condenser outlet quality when using the Shah correlation and, therefore, the inventory prediction is smaller. This analysis clearly shows the sensitivity of the inventory prediction to the degrees of subcooling present.

6.3b Solution Technique

The following is a comparison of the error incorporated into the predicted charge determination for the two integral solution techniques (numerical average and Simpson's rule). The homogeneous void fraction correlation was used for this comparison since the closed form solution was available. The closed form solution obtained for the homogeneous void fraction correlation was considered to be the exact solution and the average error for the other techniques was calculated from this. Table 6.1 provides the results obtained for this comparison.

Inventory Numerical Methods Comparison Percent Error From Closed Form Solution		
Number of Modules	Numerical Average	Simpson's Rule
28	0.07260 %	0.00098 %
42	0.04956 %	0.00007 %
70	0.03200 %	0.00006 %

Table 6.1 - Numerical Methods Comparison

For the Simpson's rule numerical integration, twenty uniform steps were taken from the inlet to the outlet quality of the module. Results are provided for simulation runs using 28, 42 and 70 modules in the condenser.

The results indicated that both the numerical average and the Simpson's rule techniques provide adequate solutions for the predicted charge of this condenser coil. Additionally, the results show that the error for both integration solutions decreases as the number of modules in the condenser increases. For the numerical average technique, as the number of modules in the condenser increases, the difference between the inlet and outlet quality decreases which should decrease the error in the charge prediction. The same reasoning applies to the results obtained using Simpson's rule. Overall, the error for the numerical average solution was substantially higher than that for the Simpson's rule solution. This is not a problem for this coil due to the large number of modules used in the modeling, however, if the entire condensing region was modeled with one module, the Simpson's rule solution should be used. The numerical average solutions were used for the inventory results presented in this chapter.

6.3c Void Fraction Model Comparison

The following comparison is of the predicted condenser inventory for the available void fraction correlations. All inventory results were obtained using the search parameters found in Chapter 5 for the heat transfer and pressure drop correlations. This comparison is provided in Figure 6.9. Overall, the results indicate that the Hughmark correlation consistently predicted the greatest charge and the next highest predictor was the Premoli correlation. The Tandon, Domanski and homogeneous correlations predicted inventories of comparable values.

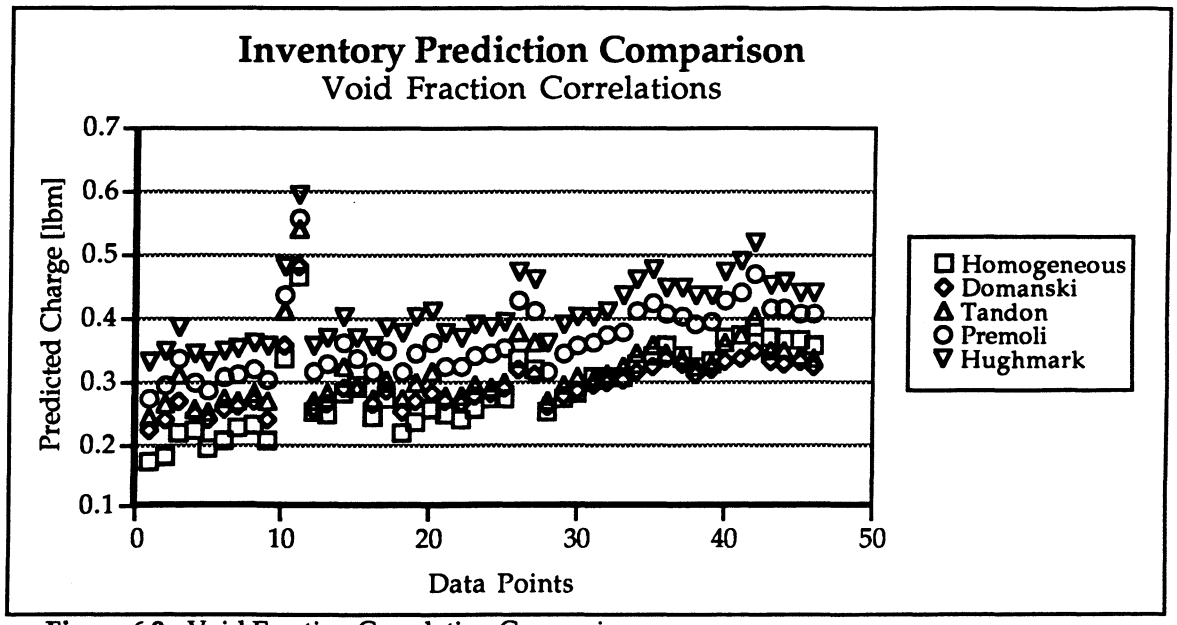

Figure 6.9 - Void Fraction Correlation Comparison

To determine which condenser inlet variables had the greatest influence on the predicted inventory, a comparison of the charge with all the inlet variables was made. The only variable showing an influence on the charge prediction was the inlet refrigerant pressure. This comparison is provided in Figure 6.10. This graph indicates that the correlations tend to predict more comparably as the inlet refrigerant pressure increases.

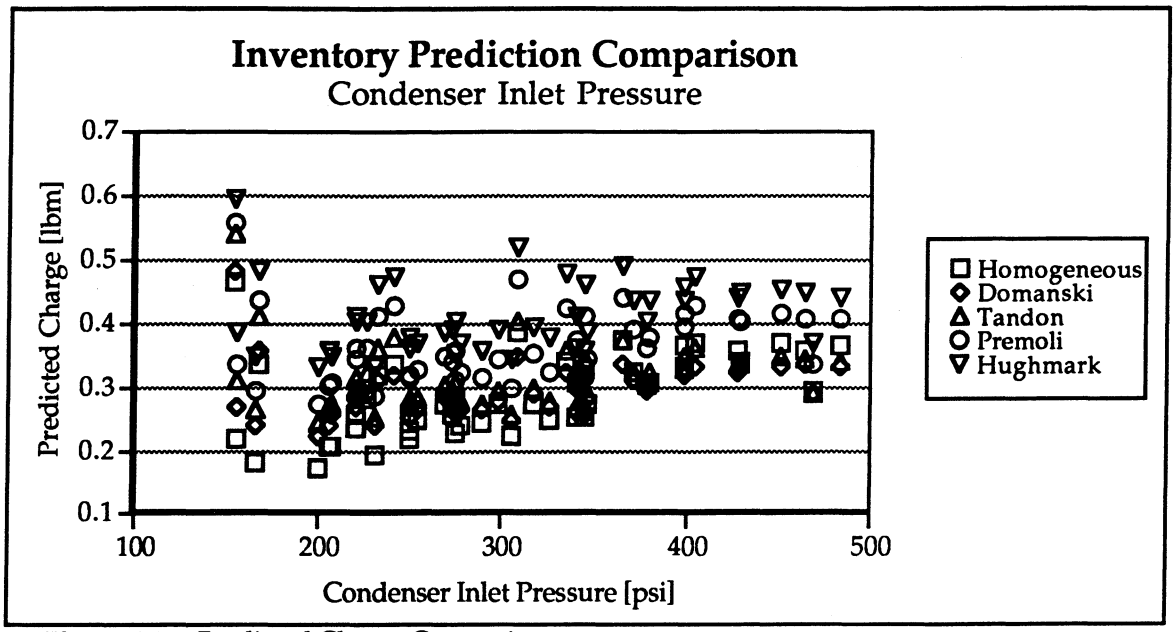

Figure 6.10 - Predicted Charge Comparison

6.3d Inventory Distribution

This last section of results provides a comparison of the mass distribution along the condenser length for the module total mass and the module liquid and vapor masses. This comparison was based on the results for a single tube. For example, this coil contained four tubes in the first manifold section but only the mass of the modules in one of these tubes was compared. This provided a clearer picture of the effect that the quality distribution has on the predicted module charge and also the effect of the refrigerant mass flux transitions. There are two mass flux transitions within this condenser coil. The first occurs between the second and third manifold sections where the number of refrigerant tubes is reduced from four to three. The second transition occurs between the sixth and seventh manifold sections where the number of refrigerant tubes is reduced from three to two. It was desired to see what effect, if any, these mass flux transitions had on the predicted inventory.

The mass distribution comparison was made for three data points at different air mass flow rates. The data points selected maintained a fairly constant refrigerant mass flow rate while varying the air mass flow rate. The data points are four, five and six from Appendix B1. In Figures 6.11 through 6.13, a comparison of the total module mass distribution along the length of the condenser coil for all five void fraction correlations is provided. All three figures indicate that the Hughmark void fraction correlation predicted the largest module charge and exhibited the most noticeable change at the mass flux transition points. In Chapter 4, the void fraction correlations were provided and it was mentioned that the Hughmark, Premoli and Tandon correlations were mass flux correlated and, therefore, it is not surprising that these correlations were most effected by the mass flux transitions.

Figures 6.11 through 6.13 also indicate that the module inventory prediction was influenced by the air mass flow rate. As the air mass flow rate increased the predicted charge also increased. This indicates that as the condensing in the coil occurred more rapidly, the exit quality of the refrigerant was smaller and the mass prediction was increased. For all three graphs, the first module was in the superheated region and there was no subcooling region.

Figures 6.14 through 6.16 provide a comparison of the liquid mass distribution along the length of the condenser for the void fraction correlations. These graphs indicates that the liquid distribution behaves in a similar nature to that of the total module mass distribution. This is not surprising due to the fact that most of the refrigerant mass in the condenser is liquid and not vapor.

In Figures 6.17 through 6.19, the same data points were used to provide a comparison of the module vapor mass distribution. Here, the increase in the air mass flow rate had the opposite effect that it had for the liquid distribution. As the air mass flow rate increased and the condensing became more rapid in the coil, the predicted module vapor mass decreased. In addition, the Hughmark void fraction correlation predicted the smallest vapor mass while the homogeneous void fraction correlation predicted the largest vapor mass.

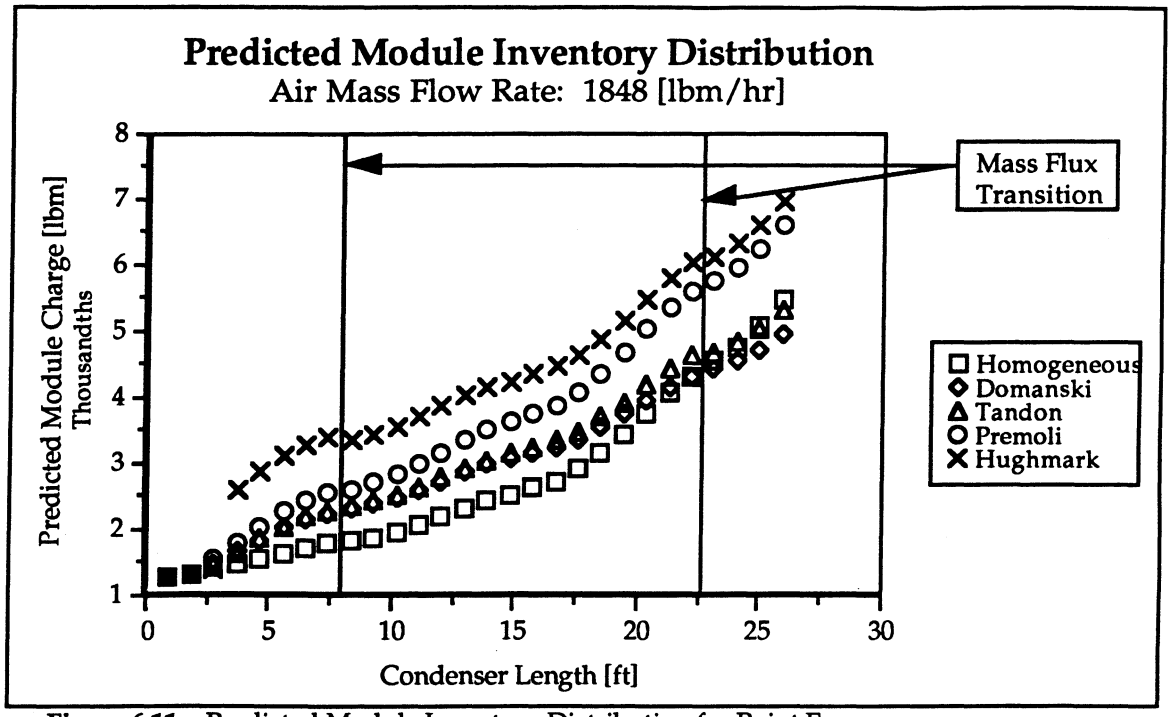

Figure 6.11 - Predicted Module Inventory Distribution for Point Four

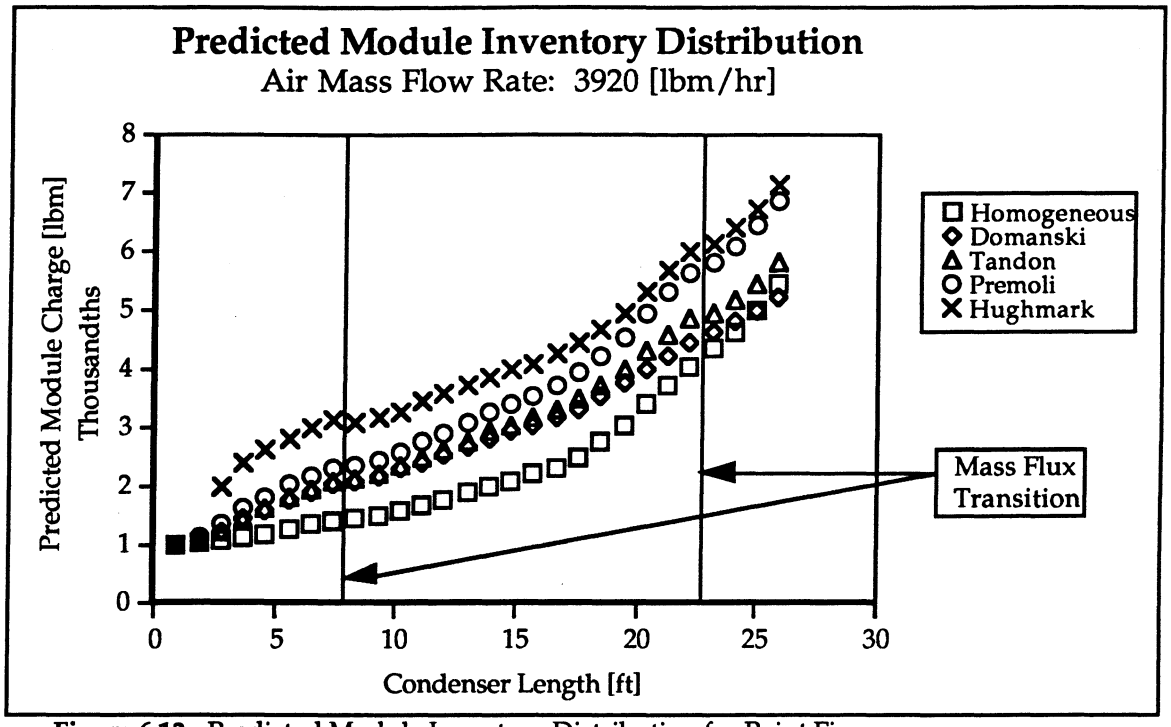

Figure 6.12 - Predicted Module Inventory Distribution for Point Five

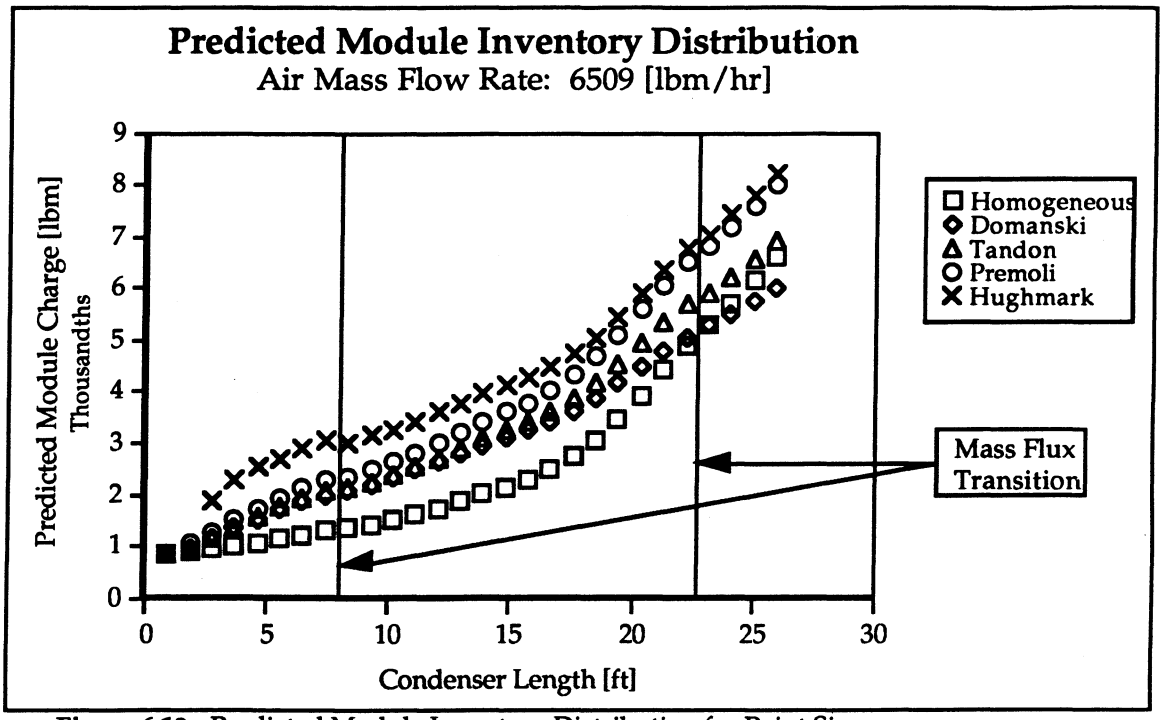

Figure 6.13 - Predicted Module Inventory Distribution for Point Six

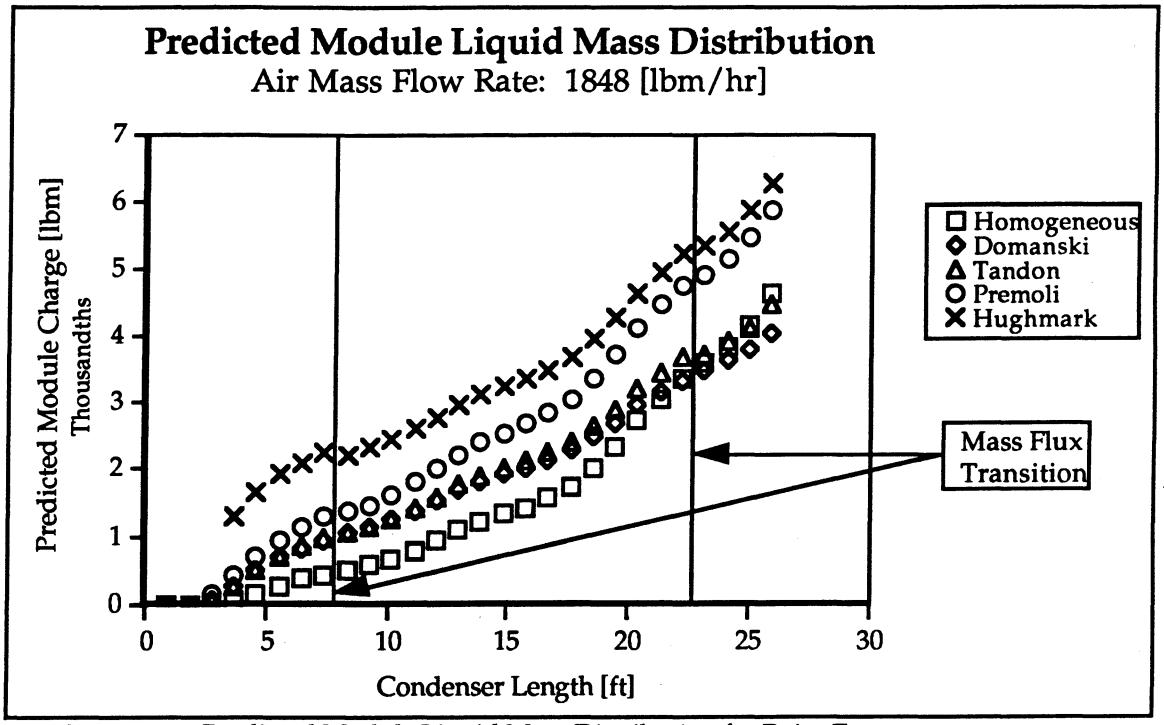

Figure 6.14 - Predicted Module Liquid Mass Distribution for Point Four

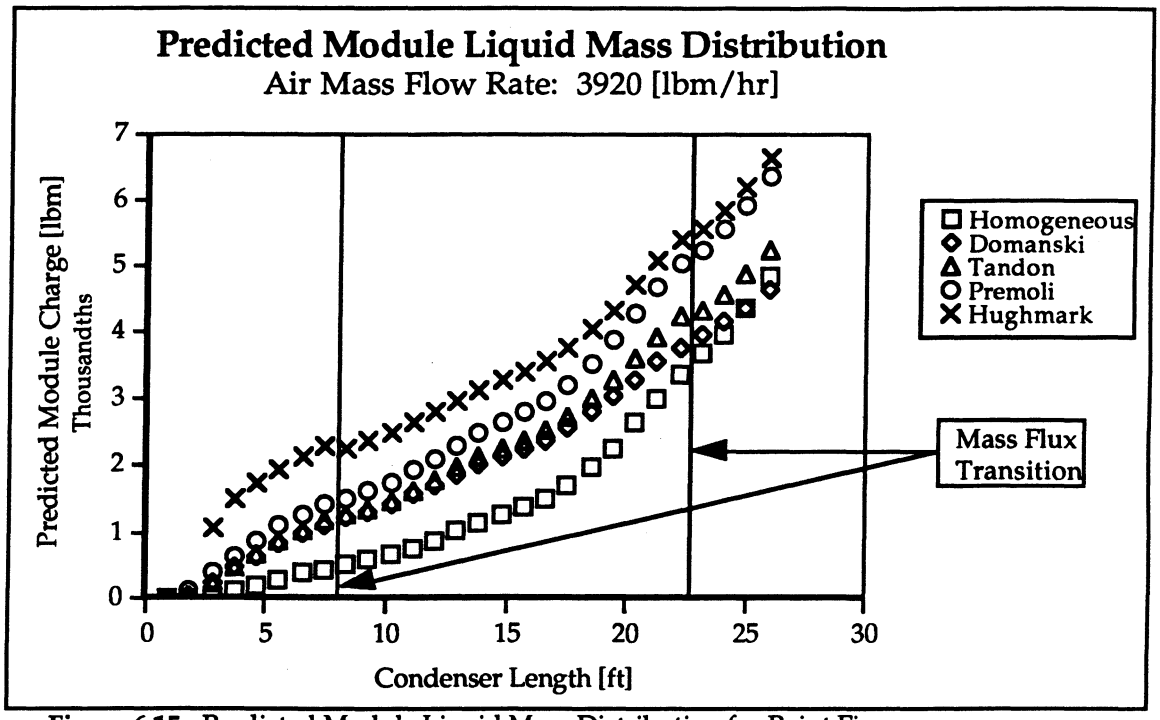

Figure 6.15 - Predicted Module Liquid Mass Distribution for Point Five

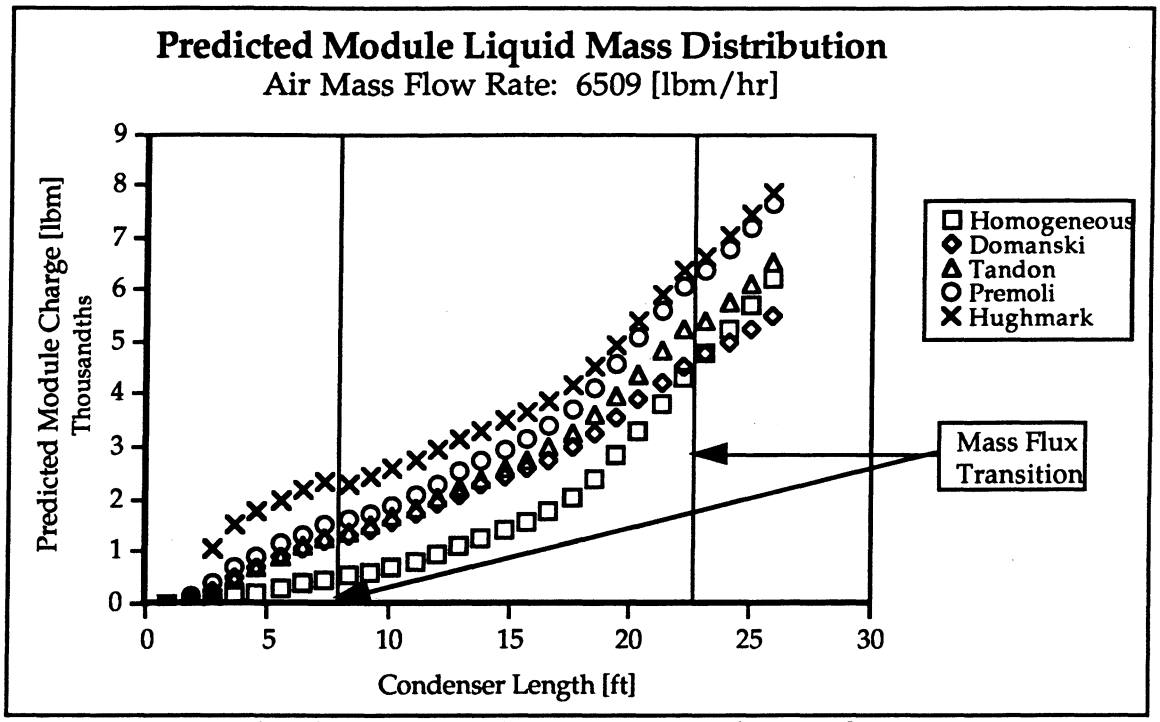

Figure 6.16 - Predicted Module Liquid Mass Distribution for Point Six

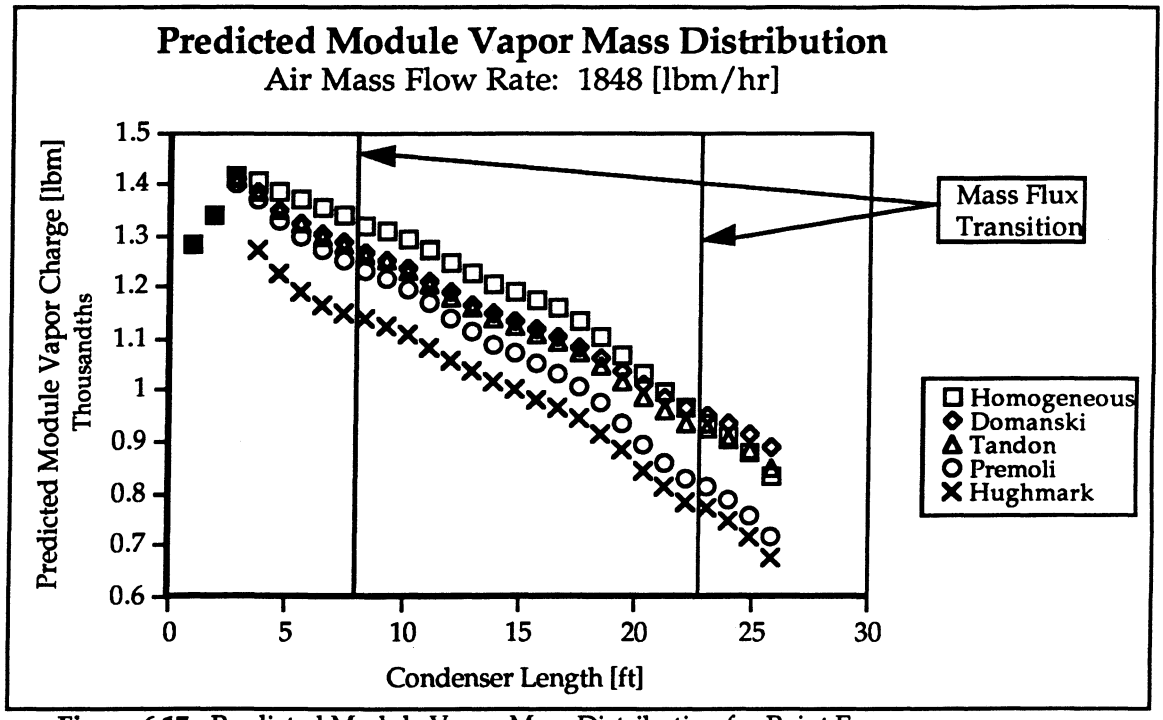

Figure 6.17 - Predicted Module Vapor Mass Distribution for Point Four

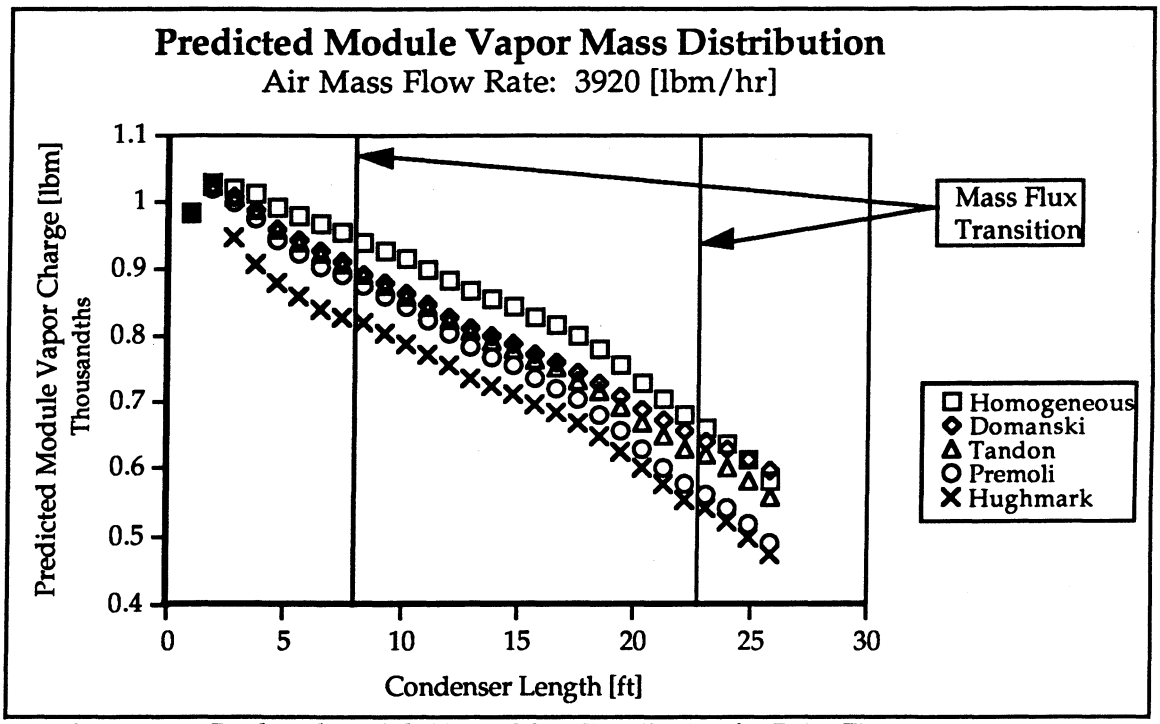

Figure 6.18 - Predicted Module Vapor Mass Distribution for Point Five

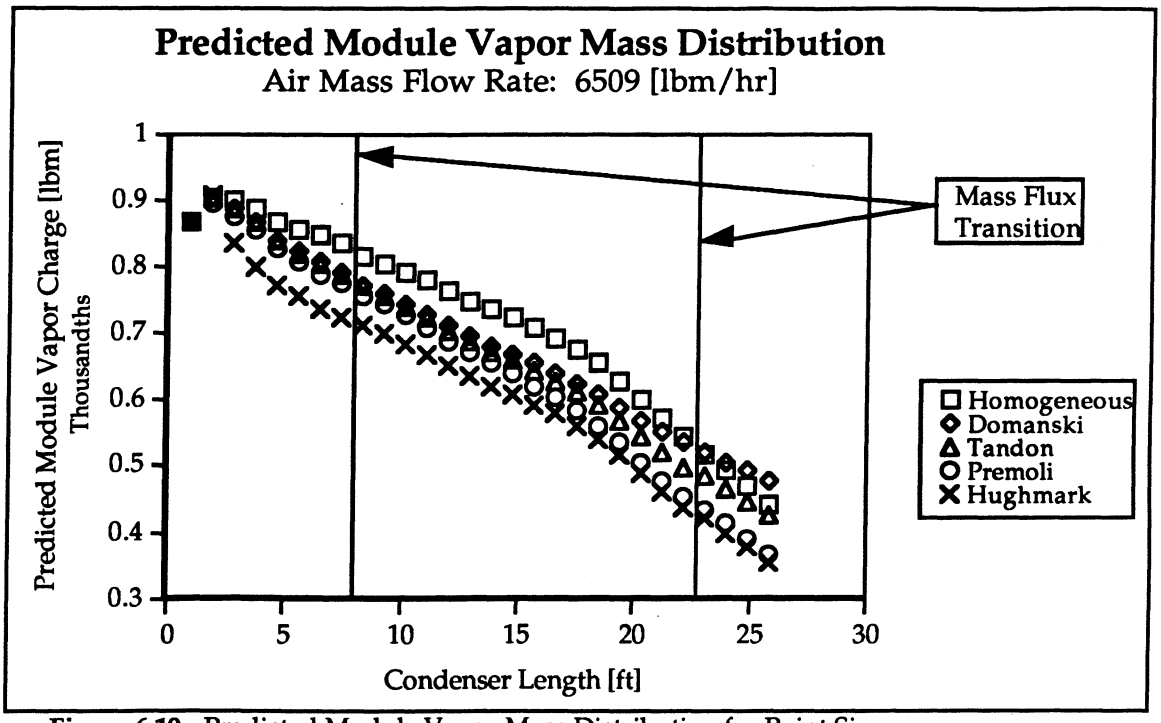

Figure 6.19 - Predicted Module Vapor Mass Distribution for Point Six

7.0 CONCLUSIONS

The Module Based Condenser Simulation program is very versatile and allows condenser coils with very complex geometries to be modeled. It provides the user with a choice of several available heat transfer coefficient, pressure drop and void fraction correlations. This makes it an extremely useful tool in studying the operation of condenser coils. The purpose of the work presented in this thesis was to determine the accuracy of the simulation program in modeling a specific condenser coil and to determine what factors most influenced the prediction of the refrigerant inventory for that coil.

Several modifications were made to the original simulation program developed by Ragazzi to allow the analysis to be performed. First, additional heat transfer coefficient correlations were added to the simulation (the Temperature Limit Principle and Dobson correlations). The Souza et.al. two-phase frictional pressure drop correlation was also added to the simulation program. Five void fraction correlations were selected from the literature to use in the calculation of the refrigerant inventory in the condenser coil. Modifications to the simulation program were made to allow for these calculations. In addition, two optimization search techniques were added to the simulation to allow for the parameter estimation process.

The results obtained and presented in Chapter 6.0, indicated that the simulation program was very accurate in it's heat capacity prediction. There was a large error in the pressure drop prediction but it is not clear what portion of that was due to experimental error. In addition there was a significant error in the simulation's prediction of the amount of subcooling at the condenser outlet. This was shown to have the greatest impact on the charge inventory calculation. Through the use of parameters estimation techniques, it was shown that the accuracy in the amount of subcooling

predicted by the simulation could be drastically improved while adding only a small amount of error to the heat capacity and pressure drop predictions.

The parameter estimation methods provide an alternative to the use of available correlations and allow the user to specify, through an objective function, the most important criteria for a specific problem. There are a vast amount of search methods which have been developed for minimization and maximization problems. The results presented here show that the selection of an appropriate search method for a specific problem is greatly influenced by the behavior of the objective function which for many real problems is unknown. In addition, this application has shown that for some problems there are many appropriate objective functions which do not always provide the same results. This makes the choice of the most appropriate objective function a critical part of the optimization process.

There are several criteria that can be used to judge the accuracy of a system component simulation program. Most frequently, the heat capacity and total pressure drop predictions are used for this determination. This work has presented an alternative criteria for judging the accuracy of a condenser simulation which is the amount of predicted subcooling. For applications which require an inventory calculation, such as for system models and transient modeling, the accuracy of the charge prediction is heavily dependent on the amount of subcooling. This suggests that for some applications the predicted subcooling of a simulation can actually be more essential than the accurate prediction of the coil heat capacity and total pressure drop.

This work has provided several insights into the problem of predicting refrigerant inventory in condenser simulation programs. There is, however, additional information which is needed for a complete understanding. First,

experimental data is required to determine the most accurate void fraction correlation available in the literature. Inventory distribution functions such as those in Figures 6.11 through 6.19 obtained with the most accurate void fraction correlation could be of significant help in analyzing and selecting condenser coil circuiting. This process may become more important as new refrigerant mixtures are put into service. Furthermore, other methods of reducing error in the Module Based Condenser Simulation program should be explored. This would include error in the methods used to determine the air side and refrigerant side heat transfer coefficient correlations and the refrigerant two phase pressure drop correlations.

REFERENCES

[1] Ragazzi, F., "Modular-Based Computer Simulation of an Air-Cooled Condenser", M.S. Thesis, University of Illinois at Urbana-Champaign, 1991.

[2] Stoecker, W.F., *Design of Thermal Systems,* 3rd ed., McGraw-Hill Publishing Co., New York, 1989, p125.

[3] Stoecker, W.F., and J.W. Jones, *Refrigeration and Air Conditioning,* 2nd ed., McGraw-Hill Publishing Co., New York, 1982, p236.

[4] Weber, R.J., "Development of an Experimental Facility to Evaluate the Performance of Air-Cooled Automotive and Household Refrigerator Condensers Utilizing Ozone-Safe Refrigerant", M.S. Thesis, University of Illinois at Urbana-Champaign, 1991, p137-144.

[5] Cavallini, A., and R. Zecchin, "A Dimensionless Correlation for Heat Transfer in Forces Convection Condensation", Heat Transfer 1974: Proceedings of the Fifth International Heat Transfer Conference, Japan Society of Mechanical Engineers, Tokyo, September 1974, Vol. III, pp309-313.

[6] Dobson, M., Ph.D. Thesis, University of Illinois at Urbana-Champaign, anticipated.

[7] McLinden, M., J. Gallagher, G. Morrison, "NIST Thermodynamic Properties of Refrigerants and Refrigerant Mixtures", U.S. Department of Commerce, National Institute of Standards and Technology, 1989.

[8] Gallagher, J., U.S.Department of Commerce, National Institute of Standards and Technology, personal communication.

[9] Rice, C.K., "The Effect of Void Fraction Correlation and Heat Flux Assumption on Refrigerant Charge Inventory Predictions", ASHRAE Transactions, Vol. 93, Part 1, 1987, pp. 341-367.

[10] Tandon, T.N., H.K. Varma, C.P. Gupta, "A Void Fraction Model for Annular Two-Phase Flow", International Journal of Heat and Mass Transfer, Vol. 28, No.1, 1985, pp. 191-198.

[11] Domanski, P., D. Didion, "Computer Modeling of the Vapor Compression Cycle with Constant Flow Area Expansion Device", NBS Building Science Series, 1983, p. 155.

[12] Premoli, A., D.D. Francesco, A. Prina, "A Dimensional Correlation for Evaluating Two-Phase Mixture Density", La Termotecnica, Vol. 25, No. 1, 1971, pp. 17-26.

[13] Kays, W., A.L. London, *Compact Heat Exchangers* , 2nd ed., McGraw Hill Publishing Co., New York, 1964.

[14] Hughmark, G.A., "Holdup in Gas-Liquid Flow", Chemical Engineering Progress, Vol. 58, No. 4, 1962, pp. 62-65.

[15] Soliman, M., J.R. Schuster, P.J. Berenson, "A General Transfer Correlation for Annular Flow Consideration", Journal of Heat Transfer, Vol. 90, 1968, pp. 267-276.

[16] Chishom, D., "A Theoretical Basis for the Lockhart-Martinelli Correlation for Two-Phase Flow", International Journal of Heat and Mass Transfer, Vol. 10, 1967, pp. 1767-1778.

[17] Zivi, S.M., "Estimation of Steady-State Steam Void Fraction by Means of the Principle of Minimum Entropy Production", Transactions ASME, Journal of Heat Transfer, Series C, Vol. 86, May 1964, pp. 247-252.

[18] Baroczy, C.J., "Correlation of Liquid Fraction in Two-Phase Flow With Application to Liquid Metals", Chemical Engineering Progress Symposium Series, Vol. 61, No. 57, 1965, pp. 179-191.

[19] Smith, S.L., "Void Fractions in Two-Phase Flow: a Correlation Based Upon an Equal Velocity Head Model", Proc. Instn. Mech. Engrs., London, Vol. 184, Part 1, No. 36, 1969, pp. 647-664.

[20] Eckels, S.J., and M.B. Pate, "A Comparison of R-134a and R-12 In Tube Heat Transfer Coefficients Based on Existing Correlations", ASHRAE Transactions, Vol. 96, Part 1, 1990.

[21] Wallis, G.B., *One-Dimensional Two-Phase Flow,* McGraw-Hill, Inc., New York, 1983, p 666.

[22] Lockhart, R.W., R.C. Martinelli, "Proposed Correlation Data for Isothermal Two-Phase Two-Component Flow in Pipes", Chemical Engineering Progress, Vol. 45, No. 1, pp. 39-48.

[23] Paliwoda, A., "Generalized Method of Pressure Drop Calculation Across Pipe Components Containing Two-Phase Flow of Refrigerants",

International Journal of Refrigeration, Vol. 15, No. 2, 1992, pp. 119-125.

[24] Marin, O., "Development of the Temperature Limit Principle for Determining Heat Transfer Characteristics of Cross Flow Heat Exchangers", M.S. Thesis, University of Illinois at Urbana-Champaign, 1993.

[25] Shah, M.M., "Heat Transfer During Film Condensation in Tubes and Annuli: a Review of the Literature", ASHRAE Transactions, Vol. 87, Part 1, 1981.

[26] Souza, A.L., et. al., "Pressure Drop During Two-Phase Flow of Refrigerants in Horizontal Smooth Tubes", ACRC Technical Report, University of Illinois, October, 1992.

[27] Rahman, M.M., Fathi, A.M. and Soliman, H.M., "Flow Pattern Boundaries During Condensation: New Experimental Data", The Canadian Journal of Chemical Engineering, Vol. 63, August, 1985.

[28] VanderZee, J., "Semi-Theoretical Steady State and Transient Modeling of a Mobile Air Conditioning Condenser", M.S. Thesis, University of Illinois at Urbana-Champaign, 1993.

APPENDIX A - SOURCE CODE

A.1 Simulation Program Source Code

```
c_____________________________________________________________________
c**********************************************************************
c************STEADY-STATE MODULE BASED CONDENSER SIMULATION*************
c******************written by Franco RAGAZZI, 1991/1992****************
c*****************converted to Fortran by Lisa ORTH 1992***************
c*********************inventory added by Lisa Orth 1992*****************

c                              NOMENCLATURE

c                            General Variables

c  TestData(,)   = Test Conditions Array
c  RunTime       = Run Time (minutes and seconds)
c  Ntests        = Total number of test runs
c  NRef$         = Refrigerant name (R12 or R134a)
c  ErrorFlag     = Keeps track of errors in the results
c  Comment       = Determines comments printed
c                    1 : 'Test_out'
c                    2 : 'Condenser_out' & 'Test_out'
c                    3 : 'Segment_out', 'Condenser_out' & 'Test_out'
c  Pchange       = Allows program to neglect or not the pressure drop
c                    1 : Pressure drop is calculated
c                    2 : Pressure drop is assumed to be zero
c
c                  Variables related to a segment analysis
c
c  SegData(,)    = Segment data input table
c  QorLSeg(,)    = Fixed length or quality segment distributions
c  NumbSeg       = Number of segments
c  SegNumb       = Segment number
c  Nmod          = Number of modules in a segment
c  Nvar          = Number of Newton-Raphson variables
c  ktube         = Condenser tube thermal conductivity [Btu/h.ft.F]
c  ModSup        = Superheat/condensing transition module number
c  ModSub        = Condensing/subcool transition module number
c  TairFactor    = Internal air temperature ratio
c  Version       = Determines which version of the simulation is run
c                    1 : Fixed length version
c                    0 : Fixed quality version
c
c                          Geometric Dimensions
c
c  Di            = Tube inside diameter                          [ft]
c  Do            = Tube outside diameter                         [ft]
c  Lsegment      = Total segment length                          [ft]
c  Ltotal        = Total condenser length                        [ft]
c  Lfrontal      = Total length of front tubes                   [ft]
c  dxdL          = Change in elevation per unit length
c  LsupSegment   = Total superheated length                      [ft]
c  LcondSegment  = Total condensing length                       [ft]
c  LsubSegment   = Total subcooled length                        [ft]
c
c                      Refrigerant related Variables
```

```
c
c  mRSegment      = Segment refrigerant mass flow rate          [lbm/hr]
c  mRtotal        = Total refrigerant mass flow rate            [lbm/hr]
c  hRiCond        = Ref. inlet enthalpy into condenser          [Bru/lbm]
c  tRiCond        = Ref. inlet temperature into condenser       [F]
c  xRiCond        = Ref. inlet quality into condenser
c  pRiCond        = Ref. inlet pressure into condenser          [psi]
c  hRiSegment     = Ref. inlet enthalpy into segment            [Btu/lbm]
c  tRiSegment     = Ref. inlet temperature into segment         [F]
c  xRiSegment     = Ref. inlet quality into segment
c  pRiSegment     = Ref. inlet pressure into segment            [psi]
c  dPfSegment     = Friction pressure change in segment         [psi]
c  dPmSegment     = Acceleration pressure change in segment     [psi]
c  dPgSegment     = Gravity pressure change in segment          [psi]
c
c                          Air related Variables
c
c  mAtotal        = Air mass flow rate over condenser           [lbm/hr]
c  mAmod          = Air mass flow rate over module              [lbm/hr]
c  tAiSegment     = Air inlet temperature into segment          [F]
c  hAiSegment     = Air inlet enthalpy into segment             [Btu/lbm]
c  pAiSegment     = Air inlet pressure into segment             [psi]
c  RHiSegment     = Air inlet relative humidity into segment
c  tAoAVGSegment  = Air segment average outlet temperature      [F]
c  tAiCond        = Air inlet temperature into condenser        [F]
c  hAiCond        = Air inlet enthalpy into condenser           [Btu/lbm]
c  pAiCond        = Air inlet pressure into condenser           [psi]
c  RHiCond        = Air inlet relative humidity into condenser
c
c                          Heat Transfer Variables
c
c  Qmod           = Heat exchanged in a module                  [Btu/hr]
c  Qsegment       = Heat exchanged in each segment              [Btu/hr]
c  RairTotal      = Total condenser air side resistance         [hr.F/Btu]
c
c  Storage Arrays:
c  "R" : refrigerant
c  "A" : air
c  "o" : module outlet
c  "i" : module inlet
c
c  pRo            = Pressure                                    [psi]
c  tRo            = Temperature                                 [F]
c  xRo            = Quality
c  hRo            = Enthalpy                                    [Btu/lbm]
c  hfo            = Saturated liquid enthalpy                   [Btu/lbm]
c  hgo            = Saturated vapor enthalpy                    [Btu/lbm]
c  vRo            = Specific volume                             [ft3/lbm]
c  vfo            = Saturated liquid specific volume            [ft3/lbm]
c  vgo            = Saturated vapor specific volume             [ft3/lbm]
c  satT           = Saturated temperature                       [F]
c  tAi            = Air inlet temperature                       [F]
c  tAo            = Air outlet temperature                      [F]
c  hAi            = Air inlet enthalpy                          [Btu/lbm]
c  hAo            = Air outlet enthalpy                         [Btu/lbm]
c  regionOUT      = Refrigerant phase (superheated, condensing or
c                                      sub-cooled)
c  Lmod           = Module length                               [ft]
```

```
c  hRef          = Refrigerant heat transfer coefficient[Btu/hr.ft.F]
c  dPfric        = Pressure change due to friction effects   [psi]
c  dPmom         = Pressure change due to acceleration effects [psi]
c  dPgrav        = Pressure change due to gravity effects    [psi]
c
c**********************************************************************
c**********************************************************************
      PROGRAM SEARCH
c
c  Exhaustive search program used for parameter estimation.
c
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION A(6),STEP(6),VALUE(25),OBJ(25),ALAST(6)
      COMMON/FUNC/OBJA,OBJB
      OPEN(UNIT=90,FILE='parameters',STATUS='old')
      OPEN(UNIT=80,FILE='step',STATUS='old')

      ITER=15
      READ(90,*) NVAR
      READ(90,*) (A(i), i=1,NVAR)
      READ(80,*) (STEP(i), i=1,NVAR)
      CLOSE(90)
      WRITE(*,*) 'Initial Parameter Values'
      WRITE(*,95) (A(i), i=1,NVAR)
      WRITE(*,*) 'Initial Step Values'
      WRITE(*,95) (STEP(i), i=1,NVAR)
      OBMIN=objective1(A)
      WRITE(*,*) 'Initial Objective Function =',OBMIN
      IQ2=0
      DO 100, i=1,ITER
         WRITE(*,*) 'Iteration =', i
         OBLAST=OBMIN
         DO 200, j=1,6
            WRITE(*,*) 'Parameter ', j
            WRITE(*,105) ' Step Size =          ',STEP(j)
            WRITE(*,*) 'Value     ', 'Objective Function'
            ALAST(j)=A(j)
            DO 300, k=-5,5
               DEL=REAL(k*STEP(j))
               A(j)=A(j)+DEL
               OB=objective1(A)
               WRITE(*,95) A(j),OB,OBJA,OBJB
               VALUE(k+6)=A(j)
               OBJ(k+6)=OB
               A(j)=A(j)-DEL
300         CONTINUE
            IQ=0
            DO 400, k=1,11
               IF (OBJ(k) .LT. OBMIN) THEN
                  OBMIN=OBJ(k)
                  A(j)=VALUE(k)
                  IQ=1
               END IF
400         CONTINUE
            WRITE(*,105) ' Objective Function = ',OBMIN
            WRITE(*,105) ' Parameter Value =    ',A(j)
200      CONTINUE
         IF (OBLAST .EQ. OBMIN) THEN
```

```
            IQ2=IQ2+1
            DO k=1,6
                STEP(k)=STEP(k)/5.0
            END DO
         END IF
         IF (IQ2 .EQ. 8) GOTO 600
100   CONTINUE
600   IF (i .EQ. ITER+1) THEN
         WRITE(*,*) 'Number of iterations =',i-1
         WRITE(*,*) 'Exited with attempted solution'
         WRITE(*,95) (A(j), j=1,NVAR)
      ELSE
         WRITE(*,*) 'Number of iterations =',i
         WRITE(*,*) 'Solution'
         WRITE(*,95) (A(j), j=1,NVAR)
      END IF
      OBMIN=objective1(A)
95    FORMAT(6(1X,F9.6))
105   FORMAT(A22,F9.6)

      END
c*************************************************************************
      FUNCTION objective2(a)

      INTEGER Ndata,Nvar,i
      REAL PARM(6),Qsearch(100,2),Qsum,dPsearch(100,2),dPsum,a(6)
      COMMON/GnrlE/PARM,Qsearch,Ndata,dPsearch
      OPEN(UNIT=90,file='parameters',status='old')

      READ(90,*) Nvar
      CLOSE(90)
      do i=1,Nvar
         PARM(i)=a(i)
      end do
      CALL Simulation
      dPsum=0.0
      do i=1,Ndata
         dPsum=dPsum+(ABS(dPSearch(i,2)-dPsearch(i,1))/dPsearch(i,2))
     *          **2.0
      end do
      objective2=((dPsum/Ndata)**0.5)

      RETURN
      END
c*************************************************************************
      FUNCTION objective1(a)

      INTEGER Ndata,Nvar,i
      REAL PARM(6),Qsearch(100,2),Qsum,dPsearch(100,2),dPsum,a(6)
      REAL dTsum,obja,objb
      COMMON/GnrlE/PARM,Qsearch,Ndata,dPsearch
      COMMON/GnrlF/dTsearch(100,2)
      COMMON/FUNC/OBJA,OBJB
      OPEN(UNIT=90,file='parameters',status='old')

      READ(90,*) Nvar
      CLOSE(90)
      do i=1,Nvar
```

```
         PARM(i)=(a(i))
       end do
       CALL Simulation
       Qsum=0.0
       dTsum=0.0
       do i=1,Ndata
          Qsum=Qsum+(ABS(Qsearch(i,2)-Qsearch(i,1))/Qsearch(i,2))**2.0
          dTsum=dTsum+(ABS(dTsearch(i,2)-dTsearch(i,1))/
     *          dTsearch(i,2))**2.0
       end do
       obja=((Qsum/Ndata)**0.5)
       objb=0.025*((dTsum/Ndata)**0.5)
       objective1=obja+objb

       RETURN
       END
c*************************************************************************
       SUBROUTINE Simulation

       IMPLICIT REAL(A-Z)
       INTEGER*4 openstat,endstat
       INCLUDE 'commain'

       TestData_in$='TestData'
       TestInfo_in$='TestInfo'
       SegmentInfo_in$='SegmentInfo'
       QorLSegment_in$='QorLSegment'
       Segment_out$='Segment_out'
       Condenser_out$='Condenser_out'
       Test_out$='Test_out'

       call ReadData
       call ClearFiles
       call GeneralPublic
       call Initial

       Ndata=Ntests
       DO 50 TestNumb=1,Ntests
         open(unit=60,file=Segment_out$,iostat=openstat,
     @                                  status='overwrite')
         write(60,*) TestNumb
         endfile(60,iostat=endstat)
         open(unit=80,file='Spread_out',status='old')
         write(80,*) TestNumb
         endfile(80,iostat=endstat)
         call TestPublic
         call Condenser

         IF (Comment.ge.2) call CondenserOutput
50     CONTINUE
       IF (Comment.ge.1) call TestOutput
       CALL CloseFiles

       RETURN
       END
c*************************************************************************
       SUBROUTINE Condenser
c
```

```
c  Routine to model a condenser made up of several segments.

      IMPLICIT REAL(A-Z)
      INTEGER i
      INCLUDE 'commain'

      DO 100 SegNumb=1,NumbSeg
c
c  Prepare data for the next segment:
c
       CALL SegmentPublic
       IF (IQref .EQ. 1) THEN
          tRiSegment=tRiCond
          pRiSegment=pRiCond
          CALL HrFpt(pRiSegment,tRiSegment,hRiSegment)
          CALL XrFph(hRiSegment,pRiSegment,xRiSegment)
       ELSE IF (IQref .EQ. 2) THEN
          tRiSegment=tRo(prevNmod)
          pRiSegment=pRo(prevNmod)
          hRiSegment=hRo(prevNmod)
          TsatiSegment=satT(prevNmod)
          vgiSegment=vgo(prevNmod)
          vfiSegment=vfo(prevNmod)
          hgiSegment=hgo(prevNmod)
          hfiSegment=hfo(prevNmod)
          regiSegment=regOUT(prevNmod)
          xRiSegment=xRo(prevNmod)
          vRiSegment=vRo(prevNmod)
       END IF
       IF (IQpd .EQ. 1) THEN
         CALL manifold(mRTotal,tRiSegment,pRiSegment,xRiSegment,
     *                 dPman)
         pRiSegment=pRiSegment-dPman
       ELSE IF (IQpd .EQ. 2) THEN
         CALL returnbend(mRsegment,tRiSegment,pRiSegment,xRiSegment,
     *                 dPret)
         pRiSegment=pRiSegment-dPret
       END IF
       IF (IQair .EQ. 1) THEN
          tAiSegment=tAiCond
       ELSE IF (IQair .EQ. 2) THEN
          tAiSegment=tAiCond+TairFactor*(tAoAVGSegment-tAiCond)
       ELSE IF (IQair .EQ. 3) THEN
          tAiSegment=tAiCond
       END IF

       RunTime=0.0
       call SegmentInitialize
       call SegmentAnalysis
       call Regime
       call Inventory
       IF (Comment.eq.3) call SegmentOutput
       IF (SegNumb.EQ.1) THEN
          Pin=pRiCond
       ELSE IF (SegNumb.EQ.NumbSeg) THEN
          pRout=pRo(Nmod)
          dPsim=Pin-pRout
          dTsub=satT(Nmod)-tRo(Nmod)
```

```
            xRout=xRo(Nmod)
          END IF

100     CONTINUE

        RETURN
        END
c*************************************************************************
        SUBROUTINE SegmentAnalysis
c
c  Routine to model a bent segment of a condenser.
c  A straight segment may be treated as a special case of the 'bent'
c  segment with TairFactor=0.0.

        IMPLICIT REAL(A-Z)
        INCLUDE 'commain'

        ErrorFlag=-1
        EPS=1.0e-04
        MAXITER=40
        call NR(Nvar,Nvar,MAXITER,0,EPS)
        call CheckResults

        IF ((ErrorFlag.ne.0).and.(Version.eq.0)) THEN
          call CorrectXro
          call NR(Nvar,Nvar,Maxiter,0,EPS)
          call CheckResults
        END IF

        RETURN
        END
c*************************************************************************
        SUBROUTINE dPcalc
c
c  Calculates the total pressure changes due to friction,
c  acceleration and elevation effects in a segment
c
        IMPLICIT REAL(A-Z)
        INTEGER i
        INCLUDE 'commain'

        dPfSegment=0.0
        dPmSegment=0.0
        dPgSegment=0.0

        DO 110 i=1,Nmod
          dPfSegment=dPfSegment + dPfric(i)
          dPmSegment=dPmSegment + dPmom(i)
          dPgSegment=dPgSegment + dPgrav(i)
110     CONTINUE

        RETURN
        END
c*************************************************************************
        SUBROUTINE CheckResults
c
c  Checks the validity of the results
c
```

```
      IMPLICIT REAL(A-Z)
      INTEGER i
      INCLUDE 'commain'

      ErrorFlag=0
c
c  ErrorFlag=1: Quality at (Nmod-1) should be zero
c  ErrorFlag=2: Qualities not in descending order
c
      IF ((xRo(Nmod).EQ.0.0).AND.(xRo(Nmod-1).NE.0.0)) ErrorFlag=1

      DO 120  i=1,Nmod
         IF (xRo(i).GT.xRo(i-1)) ErrorFlag=2
120   CONTINUE

      RETURN
      END
c*********************************************************************
      SUBROUTINE CorrectXro
c
c  Routine to correct the assigned quality distribution
c
      IMPLICIT REAL(A-Z)
      INTEGER Minm,Mod
      INCLUDE 'commain'

      IF (ErrorFlag.EQ.1) xRo(Nmod-1)=0.
c
c  Find module with lowest quality that is > xRo(Nmod)
c
      IF (ErrorFlag.EQ.2) THEN
        DO 130  Minm=1,(Nmod-1)
            IF (xRo(Minm).GT.xRo(Nmod)) goto 140
130     CONTINUE
c
c  Correct qualities that need to be corrected
c
140      IF (Nmod .NE. Minm) THEN
            dx=ABS((xRo(Minm)-xRo(Nmod))/(Nmod-Minm))
         END IF
         DO 150  Mod=(Minm+1),(Nmod-1)
            xRo(Mod)=xRo(Minm) - dx*(Mod-Minm)
150      CONTINUE
      END IF

      RETURN
      END
c*********************************************************************
      SUBROUTINE Lregions
c
c  Calculates the lengths of the super-heated,condensing
c  and sub-cooled regions in a segment
c
      IMPLICIT REAL(A-Z)
      INTEGER m,k
      INCLUDE 'commain'

      DO 160  m=1,Nmod
```

```
        IF (regOUT(m-1).NE.regOUT(m)) THEN
          IF (regOUT(m-1).EQ.1 .AND. regOUT(m).NE.1) ModSup=m
          IF (regOUT(m-1).EQ.2 .AND. regOUT(m).EQ.3) ModSub=m
        END IF
160   CONTINUE
c
c  Calculate Length of super-heating
c
      LsupSegment=0
      IF (ModSup.NE.0) THEN
        DO 170 k=1,ModSup
          LsupSegment=LsupSegment + Lmod(k)
170     CONTINUE
      END IF
c
c  Calculate Length of Sub-cooling
c
      LsubSegment=0
      IF (ModSub.NE.0) THEN
        DO 180  k=ModSub+1,Nmod
            LsubSegment=LsubSegment + Lmod(k)
180     CONTINUE
      END IF
c
c  Calculate Length of Condensing
c
      LcondSegment=Lsegment - LsubSegment - LsupSegment

      RETURN
      END
c**********************************************************************
      SUBROUTINE Xpositive
c
c  Routine to keep the elements of the 'X' array positive
c
      IMPLICIT REAL(A-Z)
      INTEGER j
      INCLUDE 'commain'

      DO 190  j=1,Nvar
        IF (X(j).LT.0.0) X(j)=X(j)*(-0.5)
190   CONTINUE

      RETURN
      END
c**********************************************************************
      SUBROUTINE HX(m)
c
c  Subroutine to simulate a general cross flow heat exchanger
c  module. (Refrigerant is unmixed, air flows over it)
c  The module supplies three residuals and assumes three
c  variables for a Newton Raphson solution
c
c  INPUTS:
c  hRo(m-1): inlet refrigerant enthalpy
c  pRo(m-1): inlet refrigerant pressure
c  tRo(m-1): inlet refrigerant temperature
c  xRo(m-1): inlet refrigerant quality
```

```
c  region:   Region that module m is in
c           (1=superheat,2=condensing & 3=subcool)
c  OUT(m-1) : module entrance
c  OUT(m)   : module exit
c  m :       module number
c  OUTPUTS:
c  hRo(m)   : outlet refrigerant enthalpy
c  hAo(m)   : outlet air enthalpy
c  pRo(m)   : outlet refrigerant pressure
c  tRo(m)   : outlet refrigerant temperature
c  xRo(m)   : outlet refrigerant quality
c  RES(,)   : Residuals for a Newton Raphson solver.
c
      IMPLICIT REAL(A-Z)
      INTEGER k,m
      INCLUDE 'commain'

      L=0.
      DO 300  k=1,Nmod
         L=L + Lmod(k)
300   CONTINUE

      mAmod=mAtotal*(Lmod(m)/Lfrontal)
      dt=(tRo(m-1)-tAi(m))
      CALL UAcalc(tRo(m-1),xRo(m-1),pRo(m-1),vgo(m-1),vfo(m-1),
     @     Lmod(m),ua,mRSegment,hRef(m))
      CALL cAIRcalc(tAi(m),mAmod,cAir)
      CALL cREFcalc(xRo(m-1),tRo(m-1),cRef,mRSegment)
      cmin=min(cAir,cRef)
      CALL EPScalc(cAir,cRef,ua,eps)
      Qmod(m)=cmin*eps*dt
      hAo(m)=Qmod(m)/mAmod + hAi(m)
      CALL TaFh(hAo(m),tAo(m))
c      write(*,*) m,regOUT(m)
      CALL MODdP(tRo(m-1),pRo(m-1),xRo(m-1),xRo(m),vRo(m-1),vRo(m),
     @           vgo(m-1),vfo(m-1),Lmod(m),dPfric(m),dPmom(m),
     @           dPgrav(m),mRSegment)
      dPmod=(ABS(dPfric(m)) - ABS(dPmom(m)) - ABS(dPgrav(m)))
      Qsegment(SegNumb)= Qsegment(SegNumb)+Qmod(m)
c
c  RESIDUALS:
c
      RES(1,m)=Qmod(m)+mRSegment*(hRo(m)-hRo(m-1))
      RES(2,m)=pRo(m-1)-pRo(m)-dPmod
      IF (regOUT(m-1).NE.regOUT(m)) THEN
         IF (regOUT(m-1).EQ.1 .AND. regOUT(m).NE.1) THEN
            ModSup=m
         ELSE IF (regOUT(m-1).EQ.2 .AND. regOUT(m).EQ.3) THEN
            ModSub=m
         END IF
      END IF

      RETURN
      END
c*********************************************************************
      SUBROUTINE PropertyUpdate
c
c  Routine to evaluate refrigerant properties
```

```
c  (enthalpy,temperature,etc)
c
      IMPLICIT REAL(A-Z)
      INTEGER k
      INCLUDE 'commain'
c
c  Q = Fixed Quality Version
c  L = Fixed Length Version
c
      IF (Version.EQ.0) THEN
         DO 350 k=1,Nmod
            CALL RegionFx(xRo(k),regOUT(k))
            CALL SatProp(pRo(k),satT(k),vgo(k),vfo(k),hgo(k),hfo(k),
     @                   hfg)
            IF (k.GT.0 .AND. k.LT.Nmod) THEN
               CALL HrFx(pRo(k),xRo(k),hgo(k),hfo(k),hRo(k))
            END IF
            CALL TrFph(pRo(k),hRo(k),tRo(k),hgo(k),hfo(k),satT(k),
     @                 regOUT(k))
            IF (regOUT(k).NE.2) THEN
               CALL VrFpt(pRo(k),tRo(k),vRo(k))
            END IF
350       CONTINUE
      ELSE IF (Version.EQ.1) THEN
         DO 360  k=0,Nmod
            CALL SatProp(pRo(k),satT(k),vgo(k),vfo(k),hgo(k),hfo(k),hfg)
            CALL RegionFph(pRo(k),hRo(k),hgo(k),hfo(k),regOUT(k))
            CALL TrFph(pRo(k),hRo(k),tRo(k),hgo(k),hfo(k),satT(k),
     @                                                 regOUT(k))
            CALL XrFph(hRo(k),pRo(k),xRo(k))
            IF (regOUT(k).NE.2) THEN
               CALL VrFpt(pRo(k),tRo(k),vRo(k))
            END IF
360      CONTINUE
      END IF

      RETURN
      END
c*********************************************************************
c  SUBROUTINES RELATED TO READING AND PRINTING:
c*********************************************************************
      SUBROUTINE CloseFiles
      IMPLICIT REAL(A-Z)
      INCLUDE 'commain'

      CLOSE(10)
      CLOSE(20)
      CLOSE(30)
      CLOSE(40)
      CLOSE(50)
      CLOSE(60)
      CLOSE(70)
      CLOSE(80)

      RETURN
      END
c*********************************************************************
      SUBROUTINE ClearFiles
```

```
c
c  Routine to clear output files.

      IMPLICIT REAL(A-Z)
      INTEGER*4 openstat,endstat
      INCLUDE 'commain'

      OPEN(unit=60, file=Segment_out$, status='overwrite')
      OPEN(unit=70, file=Condenser_out$, status='overwrite')
      OPEN(unit=50, file=Test_out$, status='overwrite')
      OPEN(unit=80, file='Spread_out', status='overwrite')

      RETURN
      END
c*********************************************************************
      SUBROUTINE ReadData
c
c  Routine to read all data from input files and store it in the
c  appropriate storage arrays.

      IMPLICIT REAL(A-Z)
      INTEGER i,j,k,Var,VarCalc,NmodMax,Nmodule
      integer*4 openstat,endstat
      INCLUDE 'commain'

      OPEN(unit=10, file=TestData_in$, status='old')
      read(10,*) Ntests
      read(10,*) NcTest
      VarCalc=2
      DO 600 k=1,Ntests
        read(10,*) (TestData(k,j), j=1,NcTest)
600   CONTINUE

      OPEN(unit=20, file=TestInfo_in$, status='old')
      read(20,*) Var
      read(20,*) (TestInfo(1,j), j=1,Var)

      OPEN(unit=30, file=SegmentInfo_in$, status='old')
      read (30,*) NumbSeg
      read (30,*) NcSeg
      VarCalc=7
      DO 650 k=1,NumbSeg
        read(30,*) (SegData(k,j), j=1,NcSeg)
650   CONTINUE

      OPEN(unit=40, file=QorLSegment_in$, status='old')
      read (40,*) NumbSeg
      NmodMax=0
      DO 700 k=1,NumbSeg
        read(40,*) Nmodule
        if (Nmodule.gt.NmodMax) NmodMax=Nmodule
        read(40,*) (QorLSeg(k,j), j=1,(Nmodule-1))
700   CONTINUE

      RETURN
      END
c*********************************************************************
      SUBROUTINE GeneralPublic
```

```
c
c  Routine to evaluate public variables that do not change for
c  a given condenser and a given set of test conditions.

      IMPLICIT REAL(A-Z)
      INCLUDE 'commain'

      Comment=      int(TestInfo(1,1))
      Pchange=      int(TestInfo(1,2))
      Refrigerant=  int(TestInfo(1,3))
      Ltotal=       TestInfo(1,4)
      Lfrontal=     TestInfo(1,5)
      dzdL=         TestInfo(1,6)
      ktube=        TestInfo(1,7)
      Di=           TestInfo(1,8)
      Do=           TestInfo(1,9)
      Di=Di/12.0
      Do=Do/12.0

      IF (Refrigerant.eq.1) NRef$='R12'
      IF (Refrigerant.eq.2) NRef$='R134a'
      IF ((Refrigerant.ne.1).and.(Refrigerant.ne.2)) NRef$='unknown'

      RETURN
      END
c*********************************************************************
      SUBROUTINE TestPublic
c
c  Routine evaluate public variables whose values depend on the test
c  conditions.

      IMPLICIT REAL(A-Z)
      INCLUDE 'commain'

      tRiCond=  TestData(TestNumb,1)
      pRiCond=  TestData(TestNumb,2)
      tAiCond=  TestData(TestNumb,3)
      pAiCond=  TestData(TestNumb,4)
      RHiCond=  TestData(TestNumb,5)
      mRtotal=  TestData(TestNumb,6)
      mAtotal=  TestData(TestNumb,7)
      Qsearch(TestNumb,2)= TestData(TestNumb,8)
      pRoCond=  TestData(TestNumb,9)
      dPsearch(TestNumb,2)=pRiCond-pRoCond
      dTsearch(TestNumb,2)=TestData(TestNumb,10)

      RETURN
      END
c*********************************************************************
      SUBROUTINE SegmentPublic
c
c  Routine to evaluate segment related public variables.

      IMPLICIT REAL(A-Z)
      INCLUDE 'commain'

      Lsegment=          SegData(SegNumb,1)
      mRSegment=         SegData(SegNumb,2)*mRtotal
```

```
      Version=          int(SegData(SegNumb,3))
      Tairfactor=       SegData(SegNumb,4)
      Nmod=             int(SegData(SegNumb,5))
      IQair=            int(SegData(SegNumb,6))
      IQref=            int(SegData(SegNumb,7))
      IQpd=             int(SegData(SegNumb,8))
      IF (SegNumb.GT.1) THEN
         prevNmod=int(SegData(SegNumb-1,5))
      END IF

      Nvar=2*Nmod+1
      ModSup=0
      ModSub=0

      RETURN
      END
c**********************************************************************
      SUBROUTINE SegmentInitialize
c
c  Routine to initialize storage arrays related to a segment analysis.

      IMPLICIT REAL(A-Z)
      INTEGER i
      INCLUDE 'commain'

c  Evaluate segment inlet conditions (Given Pri, Hri and Tai):

c  Initialize arrays:

      IF (IQref.EQ.1) THEN
         CALL HaFt(hAiSegment,tAiSegment)
         CALL HrFpt(pRiSegment,tRiSegment,hRiSegment)
         CALL SatProp(pRiSegment,TsatiSegment,vgiSegment,
     *        vfiSegment,hgiSegment,hfiSegment,hfg)
         CALL RegionFph(pRiSegment,hRiSegment,hgiSegment,
     *        hfiSegment,regiSegment)
         CALL XrFph(hRiSegment,pRiSegment,xRiSegment)
         IF (regiSegment .NE. 2) THEN
            CALL VrFpt(pRiSegment,tRiSegment,vRiSegment)
         END IF
      ELSE IF (IQref.EQ.2) THEN
         CALL HaFt(hAiSegment,tAiSegment)
         IF (xRiSegment.EQ.1.0 .OR. xRiSegment.EQ.0.0) THEN
            CALL TrFph(pRiSegment,hRiSegment,tRiSegment,hgiSegment,
     *           hfiSegment,TsatiSegment,regiSegment)
            CALL VrFpt(pRiSegment,tRiSegment,vRiSegment)
         ELSE
             CALL XrFph(hRiSegment,pRiSegment,xRiSegment)
             CALL TrFph(pRiSegment,hRiSegment,tRiSegment,hgiSegment,
     *            hfiSegment,TsatiSegment,regiSegment)
         END IF
         CALL SatProp(pRiSegment,TsatiSegment,vgiSegment,
     *        vfiSegment,hgiSegment,hfiSegment,hfg)
         CALL RegionFph(pRiSegment,hRiSegment,hgiSegment,
     *        hfiSegment,regiSegment)
      END IF

      DO 900 i=0,Nmod
```

```
          hRef(i)=        99.0
          hRo(i)=         hRiSegment
          pRo(i)=         pRiSegment
          tRo(i)=         tRiSegment
          vRo(i)=         vRiSegment
          tAi(i)=         tAiSegment
          tAo(i)=         tAiSegment
          hAi(i)=         hAiSegment
          hAo(i)=         99.0
          satT(i)=        TsatiSegment
          hgo(i)=         hgiSegment
          hfo(i)=         hfiSegment
          vgo(i)=         vgiSegment
          vfo(i)=         vfiSegment
          xRo(i)=         xRiSegment
          regOUT(i)=      regiSegment
          Lmod(i)=        Lsegment/Nmod
          Res(1,i)=       1.0
          Res(2,i)=       1.0
900     CONTINUE

        call RairSide(RairTotal)
c
c  Initialize tAi:
c

        IF (Version.eq.0) THEN

c  Intitialize xRo:
          DO 1000 i=1,(Nmod-1)
            xRo(i)=QorLSeg(SegNumb,i)
1000      CONTINUE

c  Initialize hRo:
          DO 1100 i=1,(Nmod-1)

1100      CONTINUE

c  Initialize X():
          DO 1200 i=1,Nmod
            X(i)=Lsegment/Nmod
            X(i+Nmod)=pRiSegment
1200      CONTINUE

        END IF

        IF (Version.eq.1) THEN

c  Initialize Lmod:
          DO 1400 i=1,(Nmod-1)
            Lmod(i)=QorLSeg(SegNumb,i)
1400      CONTINUE

c  Evaluate length of last module:
          Lsum=0.0
          DO 1500 i=1,(Nmod-1)
            Lsum=Lsum+Lmod(i)
1500      CONTINUE
```

```
         Lmod(Nmod)=Lsegment-Lsum

c  Initialize X():
         DO 1600 i=1,Nmod
           X(i)=hRo(i)
           X(i+Nmod)=pRo(i)
1600     CONTINUE

       END IF

c  Initialize intermediate air temperature:
       TairCenter=tAiSegment
       X(Nvar)=TairCenter

       RETURN
       END
c**********************************************************************
       SUBROUTINE SegmentOutput
c
c  Routine to print segment related output data in output file
c  'Segment_out'.

       IMPLICIT REAL(A-Z)
       integer*4 openstat,endstat
       INTEGER iterations,k
       INCLUDE 'commain'

c  Print module related storage arrays:
       open(unit=60,file=Segment_out$,status='old')
       open(unit=80,file='Spread_out',status='old')
c       write(60,*) SegNumb
c       write(60,*) Nmod

c       write(60,75) 'TRO',(tRo(k), k=0,Nmod)
c       write(60,75) 'PRO',(pRo(k), k=0,Nmod)
c       write(60,85) 'XRO',(xRo(k), k=0,Nmod)
c       write(60,75) 'HRO',(hRo(k), k=0,Nmod)
c       write(60,75) 'TAI',(tAi(k), k=0,Nmod)
c       write(60,75) 'TAO',(tAo(k), k=0,Nmod)
c       write(60,75) (regOUT(k), k=0,Nmod)
c       write(60,*) (Lmod(k), k=0,Nmod)
c       write(60,*) (hfo(k), k=0,Nmod)
c       write(60,*) (hgo(k), k=0,Nmod)
c       write(60,*) (vfo(k), k=0,Nmod)
c       write(60,*) (vgo(k), k=0,Nmod)
c       write(60,75) (satT(k), k=0,Nmod)
c       write(60,*) (hAi(k), k=0,Nmod)
c       write(60,*) (hAo(k), k=0,Nmod)
c       write(60,135) 'hRef',(hRef(k), k=0,Nmod)
c       write(60,75) 'Qmod',(Qmod(k), k=0,Nmod)
c       write(60,95) 'dPfric',(dPfric(k), k=0,Nmod)
c       write(60,95) 'dPmom',(dPmom(k), k=0,Nmod)
c       write(60,95) 'dpgrav',(dPgrav(k), k=0,Nmod)
c       write(60,75) 'Regime',(Regimout(k), k=0,Nmod)
c       write(60,75) 'Weber',(We(k), k=0,Nmod)
c       write(60,75) 'Froude',(Fr(k), k=0,Nmod)
c       write(60,135) 'ReLiq',(ReLiq(k),k=0,Nmod)
c       write(60,135) 'Reeq',(Reeq(k),k=0,Nmod)
```

```
c       write(60,*) (X(k), k=1,Nmod)
C       write(60,*) (RR(k), k=1,Nmod)
c       write(60,105) 'MassV',(MassV(k), k=0,Nmod)
c       write(60,105) 'MassL',(MassL(k), k=0,Nmod)
c       write(60,105) 'RfMass',(RfMass(k), k=0,Nmod)
c       write(60,105) 'RgMass',(RgMass(k), k=0,Nmod)
c       write(60,*) 'Segment Mass =',MassSeg(SegNumb)

75     FORMAT(A8,20(2X,F6.2))
85     FORMAT(A8,20(2X,F6.4))
95     FORMAT(A8,20(2X,F6.5))
105    FORMAT(A8,20(2X,F8.6))
135    FORMAT(A8,20(2X,F8.2))

c       write(80,115) 'TRO','PRO','XRO','HRO','TAI','TAO','QMOD',
c      *              'DPFRIC','DPMOM','DPGRAV','MASSMOD','MASSL',
c      *              'MASSV','LMOD'

c       write(80,*) SegNumb
c       do k=0,Nmod
c          write(80,125) tRo(k),pRo(k),xRo(k),hRo(k),tAi(k),tAo(k),
c      *         Qmod(k),dPfric(k),dPmom(k),dPgrav(k),RgMass(k),massL(k),
c      *         massV(k),Lmod(k)
c       end do
c       do k=1,Nmod
c          write(80,145) Lmod(k),RgMass(k)*SegData(SegNumb,2),massL(k)*
c      *         SegData(SegNumb,2),massV(k)*SegData(SegNumb,2)
c       end do

115    FORMAT(10A8,3A10,A8)
125    FORMAT(2F8.2,F8.4,4F8.2,3F8.5,3F10.6,F8.4)
145    FORMAT(F8.4,3F10.6)

       ENDFILE(60,iostat=endstat)
       ENDFILE(80,iostat=endstat)

c  Calculate and store segment related data in array 'SegData(,)'
       IF (ITER.lt.MAXITER) iterations=ITER
       IF ((ITER.gt.MAXITER).or.(ITER.eq.MAXITER)) iterations=99
       RunTime=0.0

       SegData(SegNumb,NcSeg+1)=tAoAVGSegment
       SegData(SegNumb,NcSeg+2)=pRo(Nmod)
       SegData(SegNumb,NcSeg+3)=hRo(Nmod)
       SegData(SegNumb,NcSeg+4)=Qsegment(SegNumb)
      *                         *(1.0/SegData(SegNumb,2))
       SegData(SegNumb,NcSeg+5)=iterations
       SegData(SegNumb,NcSeg+6)=ErrorFlag
       SegData(SegNumb,NcSeg+7)=RunTime/60.0

       RETURN
       END
c*************************************************************************
       SUBROUTINE CondenserOutput
c
c  Routine to print condenser related output data in output file
c  'Condenser_out'.
```

```
      IMPLICIT REAL(A-Z)
      integer*4 openstat,endstat
      INTEGER i,j,k
      INCLUDE 'commain'

      OPEN(unit=70,file=Condenser_out$,iostat=openstat,status='old')
      write (70,*) TestNumb
      DO 2000 k=1,NumbSeg
        write(70,75) (SegData(k,j), j=1,NcSeg+7)
2000  CONTINUE
75    FORMAT(4(F4.2,', '),F3.0,', ',3(F6.2,', '),F7.2,', ',F2.0,', ',
     *       F4.2,', ',F4.2)
      ENDFILE(70,iostat=endstat)

c  Calculate condenser related variables:
      Qtotal=0.0
      DO 2100 i=1,NumbSeg
        Qtotal=Qtotal+Qsegment(i)*(1.0/SegData(i,2))
2100  CONTINUE
      TestData(TestNumb,NcTest+1)=Qtotal
      Qsearch(TestNumb,1)=Qtotal
      do i=1,Nmod
         IF (regOUT(i).EQ.2) THEN
            TRsat=real(tRo(i))
         END IF
      end do
      TestData(TestNumb,NcTest+2)=dTsub
      TestData(TestNumb,NcTest+3)=dPsim
      TestData(TestNumb,NcTest+4)=xRout
      dPsearch(TestNumb,1)=dPsim
      dTsearch(TestNumb,1)=dTsub
c      write(*,*) dPsearch(TestNumb,1)
      MassTotal=0.0
      do i=1,NumbSeg
         MassTotal=MassTotal+MassSeg(i)
      end do
      TestData(TestNumb,NcTest+5)=MassTotal

      RETURN
      END
c*********************************************************************
      SUBROUTINE TestOutput
c
c  Routine to print final results for all test runs.

      IMPLICIT REAL(A-Z)
      integer*4 openstat,endstat
      INTEGER j,k
      INCLUDE 'commain'

      OPEN(unit=50,file=Test_out$,iostat=openstat,status='old')
      write(50,85) 'TICR','PICR','TICA','PICA','RH','MREF','MAIR',
     *             'QEXP','POCR','DTEXP','QTOTAL','DTSUB','DPTOT',
     *             'XROUT','MASS'
      DO 2500 k=1,Ntests
        write(50,75) (TestData(k,j), j=1,NcTest+5)
2500  CONTINUE
75    FORMAT(F6.2,F8.2,2F7.2,F6.2,F8.2,F9.2,F10.2,F8.2,F7.2,F10.2,
```

```
     *       2F7.2,2F8.4)
 85    FORMAT(A6,A8,A7,A7,A6,A8,A9,A10,A8,A7,A10,A7,A7,A8,A8)
       ENDFILE(50,iostat=endstat)

       RETURN
       END
c*************************************************************************
c     MODULE Exchanger
c*************************************************************************
c     Heat Transfer Subroutines
c*************************************************************************
      SUBROUTINE cREFcalc(x,T,cRef,mRef)
c
c  Calculates the capacity rate for the unmixed refrigerant stream.
c  Input:  T = Temperature (F)
c          x = Quality
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  cRef = Refrigerant Heat Capacity
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'NISTCOM'

      CALL SpecHeat(x,T,CpR)
      cRef=mRef*CpR
      IF (x .NE. 1. .AND. x .NE. 0.) cRef=1.0e07

      RETURN
      END
c*************************************************************************
      SUBROUTINE cAIRcalc(T,mAmod,cAir)
c
c  Calculates the capacity rate for the unmixed air stream.
c  Input:  T = Temperature (F)
c          mAmod = Mass Flow Rate of the Air
c  Output:  cAir = Air Heat Capacity
c
      IMPLICIT REAL(A-Z)

      IF (T.LE.25) T=25.
c     cp=0.23954+(3.3481e-4*T)-(1.5229e-5*T**2)+(2.5524e-7*T**3)
c     -(1.6885e-9*T**4)+(4.2404e-12*T**5)
      cp=0.243
      cAir=mAmod*cp

      RETURN
      END
c*************************************************************************
      SUBROUTINE EPScalc(cAir,cRef,ua,eps)
c
c  Calculates the cross-flow heat exchanger effectiveness.
c  Input:  cAir = Air Heat Capacity
c          cRef = Refrigerant Heat Capacity
c          ua =
c  Output:  eps = Effectiveness
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'
```

```
      cmin=MIN(cAir,cRef)
      IF (cmin .NE. 0.) THEN
         cmax=MAX(cAir,cRef)
         cratio =cmin/cmax
         ntu=ua/cmin
         IF (cratio .LT. 0.00001) THEN
            eps=1.-EXP(-ntu)
         ELSE IF (cmin .EQ. cAir) THEN
            gamma=1.-EXP(-ntu*cratio)
            eps=(1.-EXP(-gamma/cratio))
         ELSE IF (cmin .EQ. cRef) THEN
            gammap=1.-EXP(-ntu)
            eps = (1.-EXP(-gammap*cratio))/cratio
         END IF
      ELSE
         eps=1.
      END IF

      RETURN
      END
c**********************************************************************
      SUBROUTINE UAcalc(T,x,pR,vg,vf,Lmod,UA,mRef,hRef)
c
c  Calculates the UA value for the module.
c  Input:  T = Refrigerant Temperature
c          x = Quality
c          pR = Refrigerant Inlet Pressure
c          vg = Specific Volume of the Refrigerant Vapor
c          vf = Specific Volume of the Refrigerant Liquid
c          Lmod = Length of the Module
c          mRef = Refrigerant Mass Flow Rate
c  Output:  UA =
c           hRef = Refrigerant Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Ai=PI*Di*Lmod
      Ao=PI*Do*Lmod
      Am=(Ai+Ao)/2.

c  Calculate refrigerant side resistance of the module
      CALL hRefside(T,x,pR,vg,vf,hRef,mRef)
      Rref=1./(hRef*Ai)

c  Calculate tube resistance of the module
      Rtube=(Do-Di)/(2.*ktube*Am)

c  Calculate air side resistance of the entire module
      RairModule=(RairTotal/Lmod)*Ltotal

      UA=1./(Rref+Rtube+RairModule)

      RETURN
      END
c**********************************************************************
      SUBROUTINE hRefside(T,x,pR,vg,vf,hRef,mRef)
c
```

```
c  Calculates the refrigerant side heat transfer coefficient.
c  Input:  T = Refrigerant Temperature
c          x = Quality
c          pR = Refrigerant Inlet Pressure
c          vg = Specific Volume of the Refrigerant Vapor
c          vf = Specific Volume of the Refrigerant Liquid
c          mRef = Refrigerant Mass Flow Rate
c  Output:  hRef = Refrigerant Side Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      IF (x .EQ. 1. .OR. x .EQ. 0.) THEN
         CALL hSPDittus(T,x,hRef,mRef)
      ELSE IF (x .NE. 1. .AND. x. NE. 0.) THEN
c         CALL hTPCavallini(T,x,vg,vf,hRef,mRef)
c         CALL hTPShah(T,x,pR,hRef,mRef)
         CALL hTP01(T,x,pR,hRef,mRef,vg,vf)
      END IF

      RETURN
      END
c*******************************************************************************
      SUBROUTINE RairSide(Rair)
c
c  Calculates the air side heat transfer resistance.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

c  Calculate air Reynolds number
      Dh=0.0073
      Ao=1.33
      muAir=0.045
      ReAir=(mAtotal*Dh)/(Ao*muAir)

c  Calculate air Prandtl number
      kAir=0.016
      CpAir=0.243
      PrAir=0.71

c  Calculate J factor
c      jf=0.165588*ReAir**(-0.401757)
      cst1=PARM(1)
      cst2=PARM(2)
      jf=cst1*ReAir**(-cst2)

c  Calculate ho
      Grair=mAtotal/Ao
      ho=(Grair*CpAir*jf)/(PrAir**0.6667)
c      Nud=0.055*(ReAir**0.918)*(PrAir**0.4)
c      ho=kAir*Nud/Dh

c  Calculate air side efficiency
      delta=0.005/12.
      kFin=110.
      mm=((2.0*ho)/(kFin*delta))**0.5
      LL=0.206/12.
```

```
      mmll=TANH(mm*LL)
      Nf=mmll/(mm*LL)
      Afin=10707.0/144.0
      Atotal=11385.25/144.0
      Ns=1.-((Afin/Atotal)*(1.-Nf))

c  Calculate total air side resistance
      Rair=1./(Ns*ho*Atotal)

      RETURN
      END
c*************************************************************************
c  Single-Phase Heat Transfer Coefficient Subroutines
c*************************************************************************
      SUBROUTINE hSPDittus(T,x,hSP,mRef)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  "Dittus-Boelter" single phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hSP = Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL Reynolds(T,x,ReLiq,ReVap,mRef)
      IF (x .EQ. 0.) Re=ReLiq
      IF (x .EQ. 1.) Re=ReVap
      CALL Prandtl(T,x,Pr)
      n=0.4
      CALL thermcon(x,T,k)
      Nu=0.023*(Re**0.8)*(Pr**n)
      hSP=Nu*k/Di

      RETURN
      END
c*************************************************************************
      SUBROUTINE hSPPetukhov(T,x,hSP,mRef)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  "Petukhov" single phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hSP = Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL Reynolds(T,x,ReLiq,ReVap,mRef)
      IF (x .EQ. 0.) Re=ReLiq
      IF (x .EQ. 1.) Re=ReVap
      CALL Prandtl(T,x,Pr)
      CALL thermcon(x,T,k)
      ff=(1.82*(LOG(Re)/LOG(10.)) - 1.64)**(-2.)
      Nu=((ff/8.)*Re*Pr)/(1.07+12.7*((ff/8)**0.5)*((Pr**0.67)-1.))
      hSP=Nu*k/Di
```

```
      RETURN
      END
c*********************************************************************
c  Two-Phase Heat Transfer Coefficient Subroutines
c*********************************************************************
      SUBROUTINE hTPCavallini(T,x,vg,vf,hTP,mRef)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  "Cavallini-Zecchin" two-phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          vg = Specific Volume of the Refrigerant Vapor
c          vf = Specific Volume of the Refrigerant Liquid
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hTP = Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL Reynolds(T,x,ReLiq,ReVap,mRef)
      CALL viscosity(0.,T,muL)
      CALL viscosity(1.,T,muG)
      Reeq=ReLiq + ReVap*((muG/muL)*((vg/vf)**0.5))
      CALL Prandtl(T,0.,PrL)
      CALL thermcon(0.,T,kL)
      CST3=PARM(3)
      CST4=PARM(4)
      hTP=CST3*(Reeq**CST4)*(PrL**0.33)*(kL/Di)
c      hTP=0.05*(Reeq**0.8)*(PrL**0.33)*(kL/Di)

      RETURN
      END
c*********************************************************************
      SUBROUTINE hTPTraviss(T,x,pR,vRR,vg,vf,hTP,mRef)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  "Traviss" two-phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          pR = Refrigerant Inlet Pressure
c          vRR = Specific Volume of the Refrigerant Mixture
c               vRR=vf+(x*(vg-vf))
c          vg = Specific Volume of the Refrigerant Vapor
c          vf = Specific Volume of the Refrigerant Liquid
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hTP = Heat Transfer Coefficient
c
      IMPLICIT REAL(A-Z)
      CHARACTER flow$*2
      INCLUDE 'comexch'

      CALL Reynolds(T,x,ReLiq,ReVap,mRef)
      CALL Prandtl(T,0.,PrL)
      CALL FanningdP(T,pR,x,vRR,vg,vf,dPdZfLiq,dPdZfVap,flow$)
      Xtt=(dPdZfLiq/dPdZfVap)**0.5
      Ftt=0.15*(1/Xtt + 2.85*(Xtt**(-0.476)))
c  Calculate F2
```

```
      IF (ReLiq .LT. 50.) THEN
         F2=0.707*PrL*(ReLiq**0.5)
      ELSE IF (ReLiq .GT. 50. .AND. ReLiq .LT. 1125.) THEN
         F2=5.*PrL + 5.*LOG(1.+ PrL*(0.09636*(ReLiq**0.585)-1.))
      ELSE IF (ReLiq .GT. 1125.) THEN
         F2=5.*PrL+5.*LOG(1.+5.*PrL)+2.5*LOG(0.00313*(ReLiq**0.812))
      END IF
      Nu=(Ftt/F2)*(PrL*(ReLiq**0.9))
      CALL thermcon(0.,T,kL)
      hTP=Nu*kL/Di

      RETURN
      END
c**********************************************************************
      SUBROUTINE hTPShah(T,x,pR,hTP,mRef)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  "Shah" two-phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          pR = Refrigerant Inlet Pressure
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hTP = Heat Transfer Coefficient
c
c  NOTE:  Use only for R12, do not have Pcrit for R134a
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Pcrit=CRIT(4,16)
      CALL viscosity(0.,T,muL)
c  Note form of Reliq
      ReLiq=(4*mRef)/(PI*Di*muL)
      Prd=pR/Pcrit
      CALL Prandtl(T,0.,PrL)
      n=0.4
      CALL thermcon(0.,T,kL)
      hl=0.023*(ReLiq**0.8)*(PrL**n)*(kL/Di)
      hl=hl*((1.-x)**0.8)
      Z=(Prd**0.4)*(((1./x)-1.)**0.8)
      psi=1. + (3.8/(Z**0.95))
      hTP=hl*psi

      RETURN
      END
c**********************************************************************
      SUBROUTINE hTP01(T,x,pR,hTP,mRef,vgi,vfi)
c
c  Calculates the refrigerant heat transfer coefficient using the
c  Dobson two-phase correlation.
c  Input:  T = Temperature (F)
c          x = Quality
c          pR = Refrigerant Inlet Pressure
c          mRef = Mass Flow Rate of the Refrigerant
c  Ouput:  hTP = Heat Transfer Coefficient
c
c
      IMPLICIT REAL(A-Z)
```

```
      INCLUDE 'comexch'

      Area=(pi*Di*Di)/4.0
      G=mRef/Area
      CALL Viscosity(1.0,T,Mug)
      CALL Viscosity(0.0,T,Muf)
      CALL thermcon(0.,T,kL)
      CALL Prandtl(T,0.,PrL)
      rhog=1.0/vgi
      rhof=1.0/vfi
      PI1=(rhog/rhof)**0.5
      PI2=PI1*(Muf/Mug)**0.125
      Xtt=PI2*((1.0-x)/x)**0.875
      ReL=G*Di*(1.0-x)/Muf
      hl=0.023*kL*ReL**0.8*PrL**0.3/Di
      cst3=PARM(3)
      cst4=PARM(4)
      hTP=cst3*hl/Xtt**cst4
c      hTP=2.61*hl/Xtt**0.80

      RETURN
      END
c*****************************************************************
c  Pressure Change Subroutines
c*****************************************************************
      SUBROUTINE MODdP(tRi,pRi,xRi,xRo,vRi,vRo,vgi,vfi,Lmod,dPfric,
     @ dPmom,dPgrav,mRef)
c
c  This routine checks what region the module is in, in order to call
c  the appropriate pressure change routine (two or single phase).
c  The public variable 'Pchange' can be used to skip the pressure
c  change calculations.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      IF (Pchange .EQ. 1) THEN
         IF (xRi .EQ. 1.0 .OR. xRi .EQ. 0.0) THEN
            CALL OnePhasedP(tRi,pRi,xRi,vRi,vRo,vgi,vfi,Lmod,
     @           dPfric,dPmom,dPgrav,mRef)
         ELSE
c            CALL TWOPhasedP(tRi,pRi,xRi,xRo,vRi,vgi,vfi,Lmod,
c     @           dPfric,dPmom,dPgrav,mRef)
            CALL TWOPhasedP01(tRi,pRi,xRi,xRo,vRi,vgi,vfi,
     @           Lmod,dPfric,dPmom,dPgrav,mRef)
         END IF
      END IF
      RETURN
      END
c*****************************************************************
      SUBROUTINE ONEPhasedP(tRi,pRi,xRi,vRi,vRo,vgi,vfi,Lmod,dPfric,
     @ dPmom,dPgrav,mRef)
c
c  Single-phase pressure change calculation.
c
c  Friction Pchange
c
      IMPLICIT REAL(A-Z)
```

```
      CHARACTER flow$*2
      INCLUDE 'comexch'

      CALL FanningdP(tRi,pRi,xRi,vRi,vgi,vfi,dPdZfLiq,dPdZfVap,
     @ flow$,mRef)

      IF (xRi .EQ. 0.0) dPdZfrict=dPdZfLiq
      IF (xRi .EQ. 1.0) dPdZfrict=dPdZfVap
c     write(*,*) 'dPFrict = ',dPdZfrict
c
c     Area=(pi*Di*Di)/4
c     G=mRef/Area
c
c  Momentum Pchange
      dPdzmom=0.0
c     dPdZmom=G*G*(vRi-vRo)/Lmod
c  Units conversion
c     dPdZmom=(dPdZmom)/(32.2*3600*3600*144)
c
c  Elevation Pchange
      CALL ElevdP(xRi,tRi,pRi,vRi,vgi,vfi,dPdZgrav)

c  TOTAL Pchange
      dPfric=dPdZfrict*Lmod
      dPmom=dPdZmom*Lmod
      dPgrav=dPdZgrav*Lmod

      RETURN
      END
c*******************************************************************************
      SUBROUTINE FanningdP(tR,pR,xR,vRR,vg,vf,dPdZfLiq,dPdZfVap,
     @  flow$,mRef)
c
c  Calculates single-phase friction pressure drop
c  using the 'Fanning' friction factor.
c
      IMPLICIT REAL(A-Z)
      CHARACTER flow$*2
      INCLUDE 'comexch'

      flow$="tt"
      CALL Reynolds(tR,xR,ReLiq,ReVap,mRef)
c      write(*,*) tR,xR,ReLiq,ReVap
c
c  **Note: t: turbulent and L: laminar
c          First letter corresponds to the liquid
c          Second one to the vapor
c
      IF (ReLiq .GT. 2300. .AND. ReVap .GT. 2300.) flow$="tt"
      IF (ReLiq .GE. 2300. .AND. ReVap .LE. 2300.) flow$="tL"
      IF (ReLiq .LE. 2300. .AND. ReVap .GE. 2300.) flow$="Lt"
      IF (ReLiq .LE. 2300. .AND. ReVap .LE. 2300.) flow$="LL"

      CALL FannFact(0.0,ReLiq,fLiq)
      CALL FannFact(1.0,ReVap,fVap)
      Area=(PI*Di*Di)/4.0
      G=mRef/Area
```

```
      IF (xR .EQ. 0.0) viLiq=vRR
      IF (xR .EQ. 1.0) viVap=vRR
      IF (xR .GT. 0.0 .AND. xR .LT. 1.0) THEN
         viLiq=vf
         viVap=vg
      END IF

      IF (xR .EQ. 0.0) THEN
         dPdZfVap=0.0
         dPdZfLiq=((2.*fLiq*G*G*viLiq)/Di)*((1-xR)**2.)
         dPdZfLiq=(dPdZfLiq)/(32.2*3600.*3600.*144.)
      ELSE IF (xR .EQ. 1.0) THEN
         dPdZfLiq=0.0
         dPdZfVap=((2.*fVap*G*G*viVap)/Di)*(xR**2.)
         dPdZfVap=(dPdZfVap)/(32.2*3600.*3600.*144.)
      ELSE
c         write(*,*) fVap,fLiq,G,viVap,viLiq
         dPdZfVap=((2.*fVap*G*G*viVap)/Di)*(xR**2.)
         dPdZfLiq=((2.*fLiq*G*G*viLiq)/Di)*((1-xR)**2.)
         dPdZfVap=(dPdZfVap)/(32.2*3600.*3600.*144.)
         dPdZfLiq=(dPdZfLiq)/(32.2*3600.*3600.*144.)

      END IF

      RETURN
      END
c*****************************************************************************
      SUBROUTINE FannFact(x,Re,ff)
c
c  Calculates 'Fanning' friction factor. Correlation used
c  depends on both the Reynolds # and the single-phase
c  region the flow is in (superheated or sub-cooled).
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      IF (Re .EQ. 0.0) ff=0.0
      IF (Re .GT. 2300.) THEN
         IF (x .EQ. 1.0) ff=0.046/(Re**(0.20))
         IF (x .EQ. 0.0) ff=0.079/(Re**(0.25))
      ELSE IF (Re.LE.2300. .AND. Re.NE.0.0) THEN
         ff=16./Re
      END IF

      RETURN
      END
c*****************************************************************************
      SUBROUTINE TWOPhasedP(tRi,pRi,xRi,xRo,vRi,vgi,vfi,Lmod,dPfric,
     @ dPmom,dPgrav,mRef)
c
c  Calculates pressure change in the two-phase region.
c
      IMPLICIT REAL(A-Z)
      CHARACTER flow$*2
      INCLUDE 'comexch'

      IF (xRo .EQ. 0.0) xRo=0.00000001
c
```

```
c  Two-phase friction pressure change
c
       CALL FanningdP(tRi,pRi,xRi,vRi,vgi,vfi,dPdZfLiq,dPdZfVap,
      @ flow$,mRef)
c       write(*,*) dPdZfLiq,dPdZfVap
       Xi=(dPdZfLiq/dPdZfVap)**0.5
       CALL PHIcalc(Xi,flow$,PHIg)
       dPdZfrict=(PHIg**2.)*dPdZfVap
c
c  Two-phase momentum Pchange
c
       CALL alphaCALC(xRi,vgi,vfi,alphai)
       CALL alphaCALC(xRo,vgi,vfi,alphao)
       CALL PmomTWOphase(tRi,pRi,vgi,vfi,xRi,xRo,alphai,alphao,
      @ Lmod,dPdZmom,mRef)
c
c  Two-phase elevation Pchange
c
       CALL ElevdP(xRi,tRi,pRi,vRi,vgi,vfi,dPdZgrav)
c
c  TOTAL two-phase Pchange
c
       dPfric=dPdZfrict*Lmod
       dPmom=dPdZmom*Lmod
       dPgrav=dPdZgrav*Lmod

       IF (xRo .EQ. 0.00000001) xRo=0.0

       RETURN
       END
c*************************************************************************
       SUBROUTINE TWOPhasedP01(tRi,pRi,xRi,xRo,vRi,vgi,vfi,Lmod,dPfric,
      @                        dPmom,dPgrav,mRef)
c
c  Evaluates the two phase region pressure drop using the Souza
c  pressure drop correlation
c
       IMPLICIT REAL(A-Z)
       INCLUDE 'comexch'

       IF (xRo .EQ. 0.0) xRo=0.000001
       Area=(pi*Di*Di)/4.0
       G=mRef/Area/3600
       rhog=1.0/vgi
       rhof=1.0/vfi
       CALL Reynolds(tRi,0.0,ReLiq,ReVap,mRef)
       CALL Viscosity(1.0,tRi,Mug)
       CALL Viscosity(0.0,tRi,Muf)
       Fr=G**2.0/(Di*32.2*rhof**2.0)
       PI1=(rhog/rhof)**0.5
       PI2=PI1*(Muf/Mug)**0.125
       flo=0.079/ReLiq**0.25
       dPlo=(2.0*flo*Lmod*G**2.0)/(rhof*Di*32.2)
       IF (xRi .GE. 0.05) THEN
          Xttin=PI2*((1.0-xRi)/xRi)**0.875
          Xttout=PI2*((1.0-xRo)/xRo)**0.875
          IF (Fr .LE. 0.7) THEN
             const1=4.172+(5.480*Fr)-(1.564*Fr**2.0)
```

```
            const2=(1.773-(0.169*Fr))
          ELSE
c             const1=7.242
c             const2=1.655
             const1=PARM(5)
             const2=PARM(6)
          END IF
          phiin=(1.376+const1*(Xttin**(-const2)))*(1.0-xRi)**1.75
          phiout=(1.376+const1*(Xttout**(-const2)))*(1.0-xRo)**1.75
          dPfric=dPlo*(phiin+phiout)/2.0/144.0
        ELSE
          dPfric=dPlo/144.0
        END IF

        alphai=1.0/(1.0+((1.0-xRi)/xRi)*PI1**0.67)
        alphao=1.0/(1.0+((1.0-xRo)/xRo)*PI1**0.67)
        Pmomi=(xRi**2.0/(rhog*alphai))+(1.0-xRi)**2.0/
       *       (rhof*(1.0-alphai))
        Pmomo=(xRo**2.0/(rhog*alphao))+(1.0-xRo)**2.0/
       *       (rhof*(1.0-alphao))
        dPmom=-G**2.0*(Pmomo-Pmomi)/144.0/32.2

        CALL ElevdP(xRi,tRi,pRi,vRi,vgi,vfi,dPdZgrav)
        dPgrav=dPdZgrav*Lmod
        IF (xRo .EQ. 0.000001) xRo=0.0


        RETURN
        END
c*************************************************************************
        SUBROUTINE alphaCALC(xR,vg,vf,alpha)
c
c  Evaluates the void fraction (alpha).
c
        IMPLICIT REAL(A-Z)

        alpha=1.0/(1.0+ (((1.0-xR)/xR)*((vf/vg)**(2.0/3.0))))

        RETURN
        END
c*************************************************************************
        SUBROUTINE PHIcalc(X,flow$,PHIg)
c
c  Evaluates the 'PHI' factor used in the Martinelli
c  friction pressure drop calculation.
c
        IMPLICIT REAL(A-Z)
        CHARACTER flow$*2

        PHIg=1.0 + 2.85*(X**0.523)

        RETURN
        END
c*************************************************************************
        SUBROUTINE PmomTWOphase(tRi,pRi,vgi,vfi,xRi,xRo,alphai,alphao,
      @   Lmod,dPdZmom,mRef)
c
c  Calculates two-phase momentum pressure change.
```

```
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL PmomIntegral(tRi,xRi,vgi,vfi,alphai,dPdZmomI)
      CALL PmomIntegral(tRi,xRo,vgi,vfi,alphao,dPdZmomO)
      G=(4.*mRef)/(PI*Di*Di)
      dPdZmom=(dPdZmomI - dPdZmomO)*(G**2)*(1./Lmod)
      dPdZmom=(dPdZmom)/(32.2*3600.*3600.*144.)

      RETURN
      END
c**********************************************************************
      SUBROUTINE PmomIntegral(tR,xR,vg,vf,alpha,dPdZmomInt)
c
c  Evaluates the integral used to evaluate two-phase
c  momentum pressure change at a given quality value.
c
      IMPLICIT REAL(A-Z)

      dPdZmomInt=(xR**2)*vg/alpha + ((1.-xR)**2.)*vf/(1.-alpha)

      RETURN
      END
c**********************************************************************
      SUBROUTINE ElevdP(xR,tR,pR,vRR,vg,vf,dPdZgrav)
c
c  Calculates the pressure change due to elevation
c  changes in elevation.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      IF (xR .EQ. 0.0 .OR. xR .EQ. 1.0)  vAv=vRR
      IF (xR .GT. 0.0 .AND. xR .LT. 1.0) vAV=vf + xR*(vg-vf)

      dPdZgrav=(1./(144.*vAV))*dzdL

      RETURN
      END
c**********************************************************************
c  Non-Dimensional Parameter Subroutines
c**********************************************************************
      SUBROUTINE ReAlpha(T,mRef,alph,ReAlph)
c
c  Calculates the Reynolds number based on the void fraction alpha for
c  use in the Hughmark correlation.
c  Input:  T = Temperature (F)
c          mRef = Refrigerant Mass Flow Rate
c          alph = void fraction
c  Output:  ReAlph = Reynolds number based on void fraction
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL viscosity(1.0,T,mug)
      CALL viscosity(0.0,T,muf)
      Area=PI*(Di**2.0)/4.0
```

```
      Gr=mRef/Area
      ReAlph=Gr*Di/(muf+alph*(mug-muf))

      RETURN
      END
c*********************************************************************
      SUBROUTINE Reynolds(T,x,ReLiq,ReVap,mRef)
c
c  Calculates the Reynolds number for both the liquid and the vapor, in
c  the cases where each one flows alone in the tube.
c  Input:  T = Temperature (F)
c          x = Quality
c          mRef = Mass Flow Rate of the Refrigerant
c  Output:  ReLiq = Reynolds number of the Liquid Refrigerant
c           ReVap = Reynolds number of the Vapor Refrigerant
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL viscosity(1.0,T,muRvap)
      CALL viscosity(0.0,T,muRliq)
      Area=PI*(Di**2.0)/4.0
      Gr=mRef/Area
      ReLiq=(Gr*Di/muRliq)*(1.0-x)
      ReVap=(Gr*Di/muRvap)*(x)

      RETURN
      END
c*********************************************************************
      SUBROUTINE Weber(T,x,mRef,WeLiq,WeVap)
c
c  Calculates the Weber number for both the liquid and vapor, in
c  the cases where each one flows alone in the tube.
c  Input:  T = Temperature (F)
c          x = Quality
c          mRef = Mass Flow Rate of the Refrigerant
c  Output:  WeLiq = Weber number of the Liquid Refrigerant
c           WeVap = Weber number of the Vapor Refrigerant
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Area=PI*(Di**2.0)/4.0
      G=mRef/Area/3600
      gc=32.174
      CALL Psat(T,P)
      CALL SatProp(P,Tsat,vg,vf,hg,hf,hfg)
      rhof=1.0/vf
      rhog=1.0/vg
      TK=(T+459.67)/1.8
      Tcrit=374.3
      Tau=1.0-(TK/Tcrit)
c
c  This correlation for the surface tension is valid for R134a only.
c
      sigma=0.0608/4.482*0.3048*Tau**1.26
      WeLiq=Di*G**2.0/sigma/rhof/gc
      WeVap=Di*G**2.0/sigma/rhog/gc
```

```
      RETURN
      END
c*********************************************************************
      SUBROUTINE Froude(T,rhof,rhog,x,mRef,Fr)
c
c  Calculates the Froude number for use in the Hughmark correlation.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Area=PI*(Di**2.0)/4.0
      Gr=mRef/Area/3600
      gc=32.174
      Beta=x/(x+(1.0-x)*(rhog/rhof))
      Fr=1.0/gc/Di*(Gr*x/Beta/rhog)**2.0

      RETURN
      END
c*********************************************************************
      SUBROUTINE Froude2(T,P,vg,vf,x,mRef,Fr)
c
c  Calculates the Froude number for use in flow regime determination.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Area=PI*(Di**2.0)/4.0
      G=mRef/Area
      rhof=1.0/vf
      rhog=1.0/vg
      CALL viscosity(1.0,T,mug)
      CALL viscosity(0.0,T,muf)
      PI1=(rhog/rhof)**0.5
      Xtt=PI1*(muf/mug)**0.1*((1-x)/x)**0.9
      phiv=1.0+1.09*Xtt**0.039
      Rels=(1.0-x)*G*Di/muf
      Ga=32.2*Di**3.0/muf**2.0*3600*3600
      IF (Rels.LE.1250.0) THEN
         Fr=0.025*Rels**1.59*(phiv/Xtt)**1.5/Ga**0.5/32.174
      ELSE IF(Rels.GT.1250.0) THEN
         Fr=1.26*Rels**1.04*(phiv/Xtt)**1.5/Ga**0.5/32.174
      END IF

      RETURN
      END
c*********************************************************************
      SUBROUTINE Weber2(T,P,vg,vf,x,mRef,We)
c
c  Subroutine to calculate the Weber number for the flow regime
c  determination.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Area=PI*(Di**2.0)/4.0
      G=mRef/Area
      rhof=1.0/vf
```

```
      rhog=1.0/vg
      CALL viscosity(1.0,T,mug)
      CALL viscosity(0.0,T,muf)
      PI1=(rhog/rhof)**0.5
      Xtt=PI1*(muf/mug)**0.1*((1-x)/x)**0.9
      phiv=1.0+1.09*Xtt**0.039
      Rels=(1.0-x)*G*Di/muf
      Revs=x*G*Di/mug
      Tk=(T+459.67)/1.8
      Tcrit=374.3
      Tau=1.0-(Tk/Tcrit)
      sigma=0.0608/4.4482*0.3048*Tau**1.26
      mug=mug/3600/32.174
      muf=muf/3600/32.174
      IF (Rels.LE.1250.0) THEN
         We=2.45*Revs**0.64*(mug**2.0/(rhog*sigma*Di))**0.3/
     *      phiv**0.4
      ELSE
         We=0.85*Revs**0.79*(mug**2.0/(rhog*sigma*Di))**0.3*
     *      ((mug/muf)**2.0*(rhof/rhog))**0.084*
     *      (Xtt/phiv**2.55)**0.039
      END IF

      RETURN
      END
c*********************************************************************
      SUBROUTINE Prandtl(T,x,Pr)
c
c  Calculates the Prandtl number for the refrigerant as a function of
c  Temperature and Quality.
c  Input:  T = Temperature (F)
c          x = Quality
c  Output:  Pr = Prandtl number
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      CALL viscosity(x,T,mu)
      CALL thermcon(x,T,k)
      CALL SpecHeat(x,T,Cp)
      Pr=(Cp*mu)/k

      RETURN
      END
c*********************************************************************
c  Thermodynamic Property Routines
c*********************************************************************
      SUBROUTINE RegionFph(p,h,hg,hf,region)

c  Determines which region the module is in, based on the refrigerant
c  enthalpy and pressure values.
c  Input:  p = pressure
c          h = enthalpy
c          hg= vapor enthalpy
c          hf= liquid enthalpy
c  Output: region= sub-cooled, condensing or superheated

      IMPLICIT REAL(A-Z)
```

```
      INCLUDE 'comexch'

      IF (h.gt.hg) region=1
      IF (h.lt.hf) region=3
      IF ((h.le.hg).and.(h.ge.hf)) region=2

      END
c*************************************************************************
      SUBROUTINE RegionFx(x,region)

c  Determines which region the module is in, based on the quality.
c  Input:  x = quality
c  Output: region= sub-cooled, condensing or superheated

      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      IF (x.eq.0.0) region=3
      IF (x.eq.1.0) region=1
      IF ((x.gt.0.0).and.(x.lt.1.0)) region=2

      END
c*************************************************************************
      SUBROUTINE TaFh(h,t)

c  Calculates the air temperature based on its enthalpy.
c  Input:  h = enthalpy [Btu/lbm]
c  Output: t = temperature [F]
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      cp=0.25
      t=(h-120.0)/cp + 68.0

      END
c*************************************************************************
      SUBROUTINE HaFt(h,t)
c
c  Calculates the air enthalpy based on its temperature.
c  Input:  h = enthalpy [Btu/lbm]
c  Output: t = temperature [F]

      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      cp=0.25
      h=(t-68.0)*cp + 120.0

      END
c*************************************************************************
      SUBROUTINE viscosity(x,T,mu)
c
c  Calculates the refrigerant viscosity as a function of temperature.
c  Input:  x = quality
c          T = Temperature [F]
c  Output:  mu = viscosity
c
```

```
      IMPLICIT REAL(A-Z)

      TRa=T + 459.67
      TK=(TRa-32.)*(5./9.) + 273.15

      IF (x .EQ. 0.) THEN
         mu=-5.9+(0.004465*TRa)+(2153/TRa)
      ELSE IF (x .EQ. 1.) THEN
         mu=0.00371+(5.714E-5*TRa)
      ELSE
         mu=0.
      END IF

      RETURN
      END
c*****************************************************************************
      SUBROUTINE thermcon(x,T,k)
c
c  Calculates thermal conductivity of the refrigerant as a function of
c  temperature.
c  Input:  x = Quality
c          T = Temperature (F)
c  Output:  k = Thermal Conductivity
c
      IMPLICIT REAL(A-Z)

      TC=(T-32.)*(5./9.)
      TR=T+459.67

      IF (x .EQ. 0.) THEN
         k=0.1039-(1.1175E-04*TR)
      ELSE IF (x .EQ. 1.) THEN
         k=0.007204-(3.199E-05*TR)+(5.866E-08*(TR**2))
      ELSE
         k=0.
      END IF

      RETURN
      END
c*****************************************************************************
      SUBROUTINE SpecHeat(x,T,Cp)
c
c  Calculates specific heat of the refrigerant as a function of
c  temperature.
c  Input:  x = Quality
c          T = Temperature (F)
c  Output:  Cp = Specific Heat
c
      IMPLICIT REAL(A-Z)
      DOUBLE PRECISION Temp
      INTEGER UTS
      INCLUDE 'NISTCOM'

      T=(T-32.0)*(5.0/9.0)
      IF (T .LT. 0.0) THEN
         T=0.0
      END IF
      IF (x.EQ.0.0) THEN
```

```
          Cp=1.3135+3.9928e-3*T+1.7497e-6*(T**2.)+4.4304e-7*(T**3.)
     &       -1.6215e-8*(T**4.)+2.0192e-10*(T**5.)
         ELSE IF (x.EQ.1.0) THEN
          Cp=0.9015+4.4663e-3*T+4.7608e-5*(T**2.)+9.9825e-7*(T**3.)
     &       -4.1929e-8*(T**4.)+4.8718e-10*(T**5.)
         ELSE
          Cp=0.0
         END IF
         T=(T*(9.0/5.0))+32.0

         RETURN
         END
c*************************************************************************
c  Property Subroutines
c*************************************************************************
         SUBROUTINE Psat(Temp,satP)
c
c  Calculates the saturation pressure of refrigerant given the
c  temperature.

         IMPLICIT REAL(A-Z)
         DOUBLE PRECISION T,P
         INTEGER UTS
         INCLUDE 'NISTCOM'

         T=dble(Temp)
         CALL PsatT(T,P)
         satP=real(P)

         RETURN
         END
c*************************************************************************
         SUBROUTINE Tsat(Press,satT)
c
c  Calculates the saturation temperature given the refrigerant pressure.

         IMPLICIT REAL(A-Z)
         DOUBLE PRECISION P,T
         INTEGER UTS
         INCLUDE 'NISTCOM'

         P=dble(Press)
         CALL TsatP(P,T)
         satT=real(T)

         RETURN
         END
c*************************************************************************
         SUBROUTINE SatProp(Press,Tsat,vg,vf,hg,hf,hfg)
c
c  Calculates the saturation properties given the refrigerant pressure.

         IMPLICIT REAL(A-Z)
         DOUBLE PRECISION Psat,T,vgg,vff,hgg,hff,hffgg,sgg,sff
         INTEGER UTS
         INCLUDE 'NISTCOM'

         Psat=dble(Press)
```

```
      CALL Saturation(Psat,T,vgg,vff,hgg,hff,hffgg,sgg,sff)
      Tsat=real(T)
      vg=real(vgg)
      vf=real(vff)
      hg=real(hgg)
      hf=real(hff)
      hfg=real(hffgg)

      RETURN
      END
c*********************************************************************
      SUBROUTINE TrFph(pR,hRR,tR,hg,hf,Tsat,region)
c
c  Calculates temperature as a function of region, enthalpy, and
c  pressure for the refrigerant stream.

      IMPLICIT REAL(A-H,O-Z)
      DOUBLE PRECISION Press,Enth,Temp,TOLH
      INTEGER UTS
      INCLUDE 'NISTCOM'

      Press=dble(pR)
      Enth=dble(hRR)
      CALL TPH(Press,Enth,Temp)
      tR=real(Temp)

      RETURN
      END
c*********************************************************************
      SUBROUTINE HrFpt(pR,tR,hRR)
c
c  Calculates refrigerant enthalpy as a function of pressure and
c  temperature.

      IMPLICIT REAL(A-Z)
      DOUBLE PRECISION Press,Temp,Enth
      INTEGER UTS
      INCLUDE 'NISTCOM'

      Press=dble(pR)
      Temp=dble(tR)
      CALL HPT(Press,Temp,Enth)
      hRR=real(Enth)

      RETURN
      END
c*********************************************************************
      SUBROUTINE HrFx(pR,xR,hg,hf,hRR)
c
c  Calculates the refrigerant enthalpy as a function of pressure and
c  quality in the two-phase region.

      IMPLICIT REAL(A-Z)
      DOUBLE PRECISION Press,Qual,Enth
      INTEGER UTS
      INCLUDE 'NISTCOM'

      Press=dble(pR)
```

```
      Qual=dble(xR)
      CALL HPX(Press,Qual,Enth)
      hRR=real(Enth)

      RETURN
      END
c*********************************************************************
      SUBROUTINE VrFpt(pR,tR,vRR)
c
c  Calculates the refrigerant specific volume as a function of pressure
c  and temperature for the subcooled and superheated regions.

      IMPLICIT REAL(A-Z)
      DOUBLE PRECISION Press,Temp,SpVol
      INTEGER UTS
      INCLUDE 'NISTCOM'

      Press=dble(pR)
      Temp=dble(tR)
      CALL VPT(Press,Temp,SpVol)
      vRR=real(SpVol)

      RETURN
      END
c*********************************************************************
      SUBROUTINE XrFph(hRo,pR,xR)
c
c  Calculates the refrigerant quality as a function of pressure and
c  enthalpy in the two-phase region.

      IMPLICIT REAL(A-Z)
      DOUBLE PRECISION Press,Enth,Qual
      INTEGER UTS
      INCLUDE 'NISTCOM'

      Press=dble(pR)
      Enth=dble(hRo)
      CALL XPH(Press,Enth,Qual)
      xR=real(Qual)

      IF (xR.GT.1.0) THEN
         xR=1.0
      ELSE IF (xR.LT.0.0) THEN
         xR=0.0
      ENDIF

      RETURN
      END
c*********************************************************************
c  Pressure Change Subroutines
c*********************************************************************
      SUBROUTINE manifold(mref,tRi,pRi,xRi,dPman)
c
c  Subroutine to calculate pressure drop in the condenser manifolds.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'
```

```
      DiSegment=Di
      Di=0.029
      Area=(pi*Di*Di)/4.0
      G=mRef/Area/3600
      CALL SatProp(pRi,Tsati,vgi,vfi,hgi,hfi,hfg)
      rhog=1.0/vgi
      rhof=1.0/vfi
      CALL Viscosity(1.0,tRi,Mug)
      CALL Viscosity(0.0,tRi,Muf)
      PI1=(rhog/rhof)
      PI2=PI1*(Muf/Mug)**0.25
      si=2.7
      const1=0.58
      Betac=(PI2+const1*(1.0-PI2)*xRi)*(1.0-xRi)**0.333+xRi**2.276
      dPman=G**2.0/(2.0*rhog)*(si*Betac)/32.2/144.0
      Di=DiSegment

      RETURN
      END
c**********************************************************************
      SUBROUTINE returnbend(mref,tRi,pRi,xRi,dPret)
c
c  Subroutine to calculate pressure drop in the condenser manifolds.
c
      IMPLICIT REAL(A-Z)
      INCLUDE 'comexch'

      Area=(pi*Di*Di)/4.0
      G=mRef/Area/3600
      CALL SatProp(pRi,Tsati,vgi,vfi,hgi,hfi,hfg)
      rhog=1.0/vgi
      rhof=1.0/vfi
      CALL Viscosity(1.0,tRi,Mug)
      CALL Viscosity(0.0,tRi,Muf)
      PI1=(rhog/rhof)
      PI2=PI1*(Muf/Mug)**0.25
      si=0.12
      const1=3.0
      Betac=(PI2+const1*(1.0-PI2)*xRi)*(1.0-xRi)**0.333+xRi**2.276
      dPret=G**2.0/(2.0*rhog)*(si*Betac)/32.2/144.0

      RETURN
      END
c**********************************************************************
      SUBROUTINE Regime
c
c  Subroutine to find the flow regime for each module.
c      1 = Annular
c      2 = Wavy
c      3 = Mist
c
      IMPLICIT REAL(A-Z)
      INTEGER i
      INCLUDE 'commain'

      do i=1,Nmod
         IF (xRo(i).NE.1.0 .AND. xRo(i).NE.0.0) THEN
            CALL Froude2(tRo(i),pRo(i),vgo(i),vfo(i),xRo(i),
```

```
     *                    mRSegment,Fr(i))
            CALL Weber2(tRo(i),pRo(i),vgo(i),vfo(i),xRo(i),
     *                  mRSegment,We(i))
            IF (We(i).GE.30.0) THEN
               Regimout(i)=3.0
            ELSE IF (We(i).LT.30.0) THEN
               IF (Fr(i).LE.7.0) THEN
                  Regimout(i)=2.0
               ELSE
                  Regimout(i)=1.0
               END IF
            END IF
         ELSE
            Fr(i)=0.0
            We(i)=0.0
            Regimout(i)=0.0
         END IF
         CALL Reynolds(tRo(i),xRo(i),ReLiq(i),ReVap(i),mRSegment)
         CALL Viscosity(1.0,tRo(i),mug)
         CALL Viscosity(0.0,tRo(i),muf)
         Reeq(i)=ReVap(i)*(mug/muf)*(vgo(i)/vfo(i))**0.5+ReLiq(i)
      END DO

      RETURN
      END
c*************************************************************************
      SUBROUTINE Inventory
c
c  Subroutine to calculate the charge in each module and the
c  overall Segment refrigerant charge.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      MassSeg(SegNumb)=0.0
      Wg(0)=1.0
      Wf(0)=0.0
      alpha(0)=1.0
      DO i=1,Nmod
         IF (xRo(i).EQ.1.0) THEN
            alpha(i)=1.0
            Wg(i)=1.0
            Wf(i)=0.0
            vRavg=(vRo(i)+vRo(i-1))/2.0
            rhoR=1.0/vRavg
            Ac=pi*(Di/2.0)**2.0
            Vmod=Ac*Lmod(i)
            RfMass(i)=Vmod*rhoR
            RgMass(i)=Vmod*rhoR
            massL(i)=0.0
            massV(i)=RgMass(i)
         ELSE IF (xRo(i).EQ.0.0 .AND. xRo(i-1).EQ.0.0) THEN
            alpha(i)=0.0
            Wg(i)=0.0
            Wf(i)=1.0
            vRavg=(vRo(i)+vRo(i-1))/2.0
            rhoR=1.0/vRavg
```

```
            Ac=pi*(Di/2.0)**2.0
            Vmod=Ac*Lmod(i)
            RfMass(i)=Vmod*rhoR
            RgMass(i)=Vmod*rhoR
            massL(i)=RgMass(i)
            massV(i)=0.0
         ELSE
            CALL Homog(i)
c            CALL HomogAvg(i)
c            CALL LockMartAvg(i)
c            CALL TandonAvg(i)
c            CALL PremoliAvg(i)
c            CALL HughmarkAvg(i)
         END IF
         RfMass(i)=RfMass(i)/SegData(SegNumb,2)
         RgMass(i)=RgMass(i)/SegData(SegNumb,2)
         massL(i)=massL(i)/SegData(SegNumb,2)
         massV(i)=massV(i)/SegData(SegNumb,2)
         MassSeg(SegNumb)=MassSeg(SegNumb)+RfMass(i)
      END DO

      RETURN
      END
c*********************************************************************
      SUBROUTINE Homog(i)
c
c  Subroutine to calculate the refrigerant inventory using the
c  closed form of the homogeneous solution.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      rhog=1.0/vgo(i)
      rhof=1.0/vfo(i)
      Ac=pi*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      a1=rhog/rhof
      b1=1.0-a1
      xo=xRo(i)
      xi=xRo(i-1)
      alpha(i)=xo/(xo+(1.0-xo)*a1)
      Wf(i)=((xo-xi)-((xo/b1)-((a1/b1**2.)*log(a1+b1*xo)))
     *      +((xi/b1)-((a1/b1**2.)*log(a1+b1*xi))))/(xo-xi)
      Wg(i)=(((xo/b1)-((a1/b1**2.)*log(a1+b1*xo)))-((xi/b1)
     *      -((a1/b1**2.)*log(a1+b1*xi))))/(xo-xi)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c*********************************************************************
      SUBROUTINE simp(t,h,tint,tfin,PI1,Wff)

      IMPLICIT REAL(A-Z)
      DIMENSION f(3)
```

```
      f(1)=(1.0-(1.0/(1.0+((1.0-t)/t)*PI1)))/(tint-tfin)
      t=t-(h/2.0)
      f(2)=(1.0-(1.0/(1.0+((1.0-t)/t)*PI1)))/(tint-tfin)
      t=t-(h/2.0)
      f(3)=(1.0-(1.0/(1.0+((1.0-t)/t)*PI1)))/(tint-tfin)
      Wff=h/6*(f(1)+4*f(2)+f(3))
      RETURN
      END
c**********************************************************************
      SUBROUTINE HomogAvg(i)
c
c  Subroutine to find the refrigerant inventory using the homogeneous
c  form with an average over the interval.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      rhog=1.0/((vgo(i)+vgo(i-1))/2.0)
      rhof=1.0/((vfo(i)+vfo(i-1))/2.0)
      Ac=PI*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      PI1=rhog/rhof
      xo=xRo(i)
      xi=xRo(i-1)
      xo=(xo+xi)/2.0
      alpha(i)=xo/(xo+(1.0-xo)*PI1)
      Wg(i)=alpha(i)
      Wf(i)=1.0-Wg(i)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c**********************************************************************
      SUBROUTINE TandonAvg(i)
c
c  Subroutine to calculate the inventory using the Tandon correlation
c  and an average alpha over the interval.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      Ac=pi*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      rhog=1.0/((vgo(i)+vgo(i-1))/2.0)
      rhof=1.0/((vfo(i)+vfo(i-1))/2.0)
      CALL viscosity(1.0,satT(i),Mug)
      CALL viscosity(0.0,satT(i),Muf)
      PI1=rhog/rhof
      PI2=PI1*(Muf/Mug)**0.2
      xo=xRo(i)
      xi=xRo(i-1)
      xo=(xo+xi)/2.0
```

```
      IF (xo.EQ.0.0) THEN
         alpha(i)=0.0
         GOTO 10
      END IF
      CALL Reynolds(satT(i),0.0,RKReL,RKReV,mRSegment)
      Xtt=(((1.0-xo)/xo)**0.9)*(PI2**0.5)
      FXtt=0.15*(1.0/Xtt + 2.85/Xtt**0.476)
      IF (RKReL.GT.50.0 .AND. RKReL.LT.1125.0) THEN
         alpha(i)=1.0-1.928*RKReL**(-0.315)/FXtt
     *           +0.9293*RKReL**(-0.63)/FXtt**2.0
      ELSE IF(RKReL.GE.1125.0) THEN
         alpha(i)=1.0-0.38*RKReL**(-0.088)/FXtt
     *           +0.0361*RKReL**(-0.176)/FXtt**2.0
      END IF
10    Wg(i)=alpha(i)
      Wf(i)=1.0-Wg(i)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c*********************************************************************
      SUBROUTINE LockMartAvg(i)
c
c  Subroutine to calculate the charge inventory using the Lockhart-
c  Martinelli correlation and an average over the interval.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      Ac=pi*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      rhog=1.0/((vgo(i)+vgo(i-1))/2.0)
      rhof=1.0/((vfo(i)+vfo(i-1))/2.0)
      CALL viscosity(1.0,satT(i),Mug)
      CALL viscosity(0.0,satT(i),Muf)
      PI1=rhog/rhof
      PI2=PI1*(Muf/Mug)**0.2
      xo=xRo(i)
      xi=xRo(i-1)
      xo=(xo+xi)/2.0
      IF (xo.EQ.0.0) THEN
         alpha(i)=0.0
         GOTO 10
      END IF
      Xtt=(((1.0-xo)/xo)**0.9)*(PI2**0.5)
      IF (Xtt.LE.10.0) THEN
         alpha(i)=(1.0+Xtt**0.8)**(-0.378)
      ELSE
         alpha(i)=0.823-0.157*log(Xtt)
      END IF
10    Wg(i)=alpha(i)
      Wf(i)=1.0-Wg(i)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
```

```
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c*****************************************************************
      SUBROUTINE HughmarkAvg(i)
c
c  Subroutine to calculate the inventory using the Hughmark correlation
c  and an average alpha over the interval.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      Ac=pi*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      rhog=1.0/vgo(i)
      rhof=1.0/vfo(i)
      PI1=rhog/rhof
      xo=xRo(i)
      xi=xRo(i-1)
      xo=(xo+xi)/2.0
      IF (xo.GE.0.990) THEN
         alpha(i)=1.0
         GOTO 20
      END IF
      alpha(i)=xo
10    CALL ReAlpha(satT(i),mRSegment,alpha(i),Re)
      CALL Froude(satT(i),rhof,rhog,xo,mRSegment,Frr)
      Beta=xo/(xo+(1.0-xo)*PI1)
      Yl=1.0-Beta
      Z=Re**0.1667*Frr**0.125/Yl**0.25
      CALL InterKh(Z,Kh)
      alph=Kh*Beta
      diff=ABS(alpha(i)-alph)
      tol=0.0001
      IF (diff.GT.tol) THEN
         alpha(i)=alph
         GOTO 10
      END IF
20    Wg(i)=alpha(i)
      Wf(i)=1.0-Wg(i)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c*****************************************************************
      SUBROUTINE InterKh(Z,Kh)
c
c  Subroutine to interpolate for Kh given Z.  Kh is needed for the
c  Hughmark correlation.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j
```

```
      DIMENSION MatZ(15),MatKh(15)
      DATA MatZ/0.0,1.3,1.5,2.0,3.0,4.0,5.0,6.0,8.0,10.0,15.0,20.0,
     *         40.0,70.0,130.0/
      DATA MatKh/0.0,0.185,0.225,0.325,0.49,0.605,0.675,0.72,0.767,
     *           0.78,0.808,0.83,0.88,0.93,0.98/

      do 100, i=1,14
         IF (Z.GE.MatZ(i) .AND. Z.LT.MatZ(i+1)) THEN
            j=i
            GOTO 200
         END IF
100   continue
200   ratio=(Z-MatZ(i))/(MatZ(i+1)-MatZ(i))
      Kh=((MatKh(i+1)-MatKh(i))*ratio)+MatKh(i)

      RETURN
      END
c**********************************************************************
      SUBROUTINE PremoliAvg(i)
c
c  Subroutine to calculate the inventory using the Premoli correlation
c  and an average alpha over the interval.
c
      IMPLICIT REAL(A-Z)
      INTEGER i,j,k
      INCLUDE 'commain'

      Ac=pi*(Di/2.0)**2.0
      V=Ac*Lmod(i)
      rhog=1.0/((vgo(i)+vgo(i-1))/2.0)
      rhof=1.0/((vfo(i)+vfo(i-1))/2.0)
      PI1=rhog/rhof
      xo=xRo(i)
      xi=xRo(i-1)
      xo=(xo+xi)/2.0
      Beta=xo/(xo+(1.0-xo)*PI1)
      y=Beta/(1.0-Beta)
      CALL Reynolds(satT(i),0.0,RKReL,RKReV,mRSegment)
      CALL Weber2(satT(i),pRo(i),vgo(i),vfo(i),0.01,mRSegment,RKWeL)
      F1=1.578*RKReL**(-0.19)*(rhof/rhog)**(0.22)
      F2=0.0273*RKWeL*RKReL**(-0.51)*(rhof/rhog)**(-0.08)
      S=1.0+F1*((y/(1.0+y*F2))-y*F2)**0.5
      alpha(i)=xo/(xo+(1.0-xo)*PI1*S)
10    Wg(i)=alpha(i)
      Wf(i)=1.0-Wg(i)
      RfMass(i)=V*(rhof*Wf(i)+rhog*(1.0-Wf(i)))
      RgMass(i)=V*(rhog*Wg(i)+rhof*(1.0-Wg(i)))
      massL(i)=V*rhof*Wf(i)
      massV(i)=V*rhog*(1.0-Wf(i))

      RETURN
      END
c**********************************************************************
      SUBROUTINE NR(NDIM,NEQ,ITLM,LDB,EPS)
C ***
C *** PERFORMS MULTIDIMENSIONAL NEWTON-RAPHSON PROCEDURE
C *** CURRENT MAXIMUM SIZE: 40 VARIABLES, 40 EQUATIONS
C *** NDIM = NUMBER OF VARIABLES
```

```
C *** NEQ = NUMBER OF EQUATIONS
C *** ITLM = MAXIMUM NUMBER OF ITERATIONS
C *** LDB = DEBUG FLAG: 0 = OFF; 1 = ON
C *** EPS = CONVERGENCE PARAMETER
C ***
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION FF(40),DX(40),FX(40,40),FXINV(40,40)
      INCLUDE 'commain'
C
C
C *** SET TOLERANCE PARAMETERS FOR LINEAR SYSTEM SOLUTION
      ABT=1.0E-05
      AVCTR=ABT
      MINIT=6
C
C *** BEGIN MAIN LOOP ************************************
C
      DO 1000 ITER=1,ITLM
C
C *** CALL RESIDUAL EVALUATION ROUTINE AND PARTIAL EVALUATION ROUTINE
C *** FOR CURRENT X VECTOR
C
      CALL PropertyUpdate
      DELTA = .01
      CALL FPRIME(NEQ,NDIM,FX,DELTA)
      CALL FVAL(NEQ,FF,RMS)
C
C *** DEBUG PRINTOUT OF COMPUTED F VECTOR AND COMPUTED FX MATRIX
C
      IF(LDB.GT.0) CALL VCWRT(NDIM,NEQ,FF,'   F ')
      IF(LDB.GT.0) CALL MXWRT(NDIM,NEQ,NEQ,FX,'  FX ')
      IF(LDB.GT.0) CALL VCWRT(NDIM,NEQ,X,'   X ')
C
C
C
C *** CALL GAUSS-JORDAN LINEAR SYSTEM ROUTINE TO COMPUTE DX VECTOR:
C ***          FX*DX = F
C
    . CALL LINSYS2(NDIM,NEQ,FX,FF,ABT,AVCTR,FXINV,DX,DET)
C
C *** UPDATE SOLUTION VECTOR X WITH CORRECTION
C *** (IF NEW X VALUE IS NEGATIVE, OUTPUT ERROR MESSAGE)
C
      DO 200 J=1,NEQ
c      WRITE(6,131) J,DX(J)
131   FORMAT(I2,2X,F14.8)
         X(J)=X(J)-DX(J)
  200 CONTINUE
C
      DO 210 J=1,NEQ
        IF (X(J).LE.0.0 .AND. Version.EQ.1) THEN
           IF (J.EQ.NVAR) THEN
           ELSE IF (X(J-1) .LE. hfo(J)) THEN
              X(J)=X(J-1)
           ELSE
              X(J)=hfo(J)
           END IF
        ELSE IF (X(J).LT.0.0) THEN
```

```
            IF (J .NE. NDIM) THEN
            WRITE (6,'(A)') ' *** WARNING - NEW X VALUE IS < 0.0'
            WRITE(*,*) 'J =',J
            CALL VCWRT(NDIM,NEQ,X,'SYSTEM VARIABLES:    ')
            RETURN
            ELSE
            X(J)=TairCenter
            END IF
          ENDIF
  210 CONTINUE

C
C
C *** PRINT DEBUG - NEW X VECTOR AND CORRECTION
C
      IF (LDB.GT.0) CALL VCWRT(NDIM,NEQ,DX,'  DX ')
C     IF (LDB.GT.0) CALL VCWRT(NDIM,NEQ,X,' XNEW')
C
C
C
C ******************  CONVERGENCE OPTION #1:  **********************
C *************  ROOT MEAN SQUARE OF RESIDUALS
******************
C
C *** NOTE: ROUTINE CURRENTLY SET UP TO HANDLE ONLY THE MAX DX TEST
C ***
C ***       TO HAVE THE CHOICE BETWEEN MAX DX TEST AND RMS TEST,
C ***       ADD PARAMETER "ICVTYP" TO ARGUMENT LIST AND SET IT IN
C ***       CALLING PROGRAM:
C ***                 ICVTYP=1    ==>   RMS TEST
C ***                 ICVTYP=2    ==>   MAX DX TEST
C ***       ALSO, UNCOMMENT THE LINES WITH C#
C
C#    IF (ICVTYP .EQ. 1) THEN
C
C#       IF (RMS .LE. EPS) THEN
C#          WRITE (3,133) ITER,RMS
C#133       FORMAT (' NWTRPH CONVERGED: ITER,RMS = ',I6,E12.4)
C#          GO TO 1010
C#       ENDIF
C
C#    ENDIF
C
C ******************  CONVERGENCE OPTION #2:  **********************
C ************************  MAX DELTA X ****************************
C
C#    IF (ICVTYP .EQ. 2) THEN
C
      DXMAX= 0.
      DO 700 I=1,NDIM
          IF (ABS(DX(I)).GT.DXMAX) DXMAX=ABS(DX(I))
C         WRITE(5,147)I,DX(I),DXMAX
147       FORMAT(I3,2(2X,F15.7))
 700  CONTINUE
C
      IF (DXMAX.LE.EPS .AND. ITER.GE.MINIT) THEN
         CALL PropertyUpdate
c         WRITE (*,155) ITER,DXMAX
```

```
c          write(*,*) (X(J),J=1,NMOD)
155       FORMAT (' NWTRPH CONVERGED: ITER, DXMAX = ',I6,E12.4)
          GO TO 1010
        ENDIF
C
C#      ENDIF
C
 1000 CONTINUE
C
C *** END MAIN LOOP ************************************
C
C       WRITE (*,21) ITER, DXMAX
   21 FORMAT (' NWTRPH NOT CONVERGED: ITER,DXMAX',I6,E12.4)
C       CALL VCWRT(NDIM,NEQ,X,'SYSTEM VARIABLES:    ')
C
 1010 CONTINUE
      RETURN
      END
C *********************************************************************
      SUBROUTINE FPRIME(NEQ,NDIM,FX,DELTA)
C ***
C *** NUMERICAL APPROXIMATION OF PARTIAL DERIVATIVE MATRIX
C ***
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION FX(40,40),FF(40),F00(40)
      INCLUDE 'commain'
C
      CALL FVAL(NEQ,F00,RMS)
      DO 20 I=1,NDIM
          DELTAX=DELTA*X(I)
          X(I)=X(I)+DELTAX
          CALL FVAL(NEQ,FF,RMS)
          DO 10 J=1,NEQ
             IF (DELTAX .EQ. 0.0) GOTO 10
             FX(J,I)=(FF(J)-F00(J))/DELTAX
   10     CONTINUE
          X(I)=X(I)-DELTAX
   20 CONTINUE

      RETURN
      END
C*********************************************************************
      SUBROUTINE FVAL(NEQ,FF,RMS)
      IMPLICIT REAL (A-Z)
      INTEGER i,j,k
      DIMENSION FF(40)
      INCLUDE 'commain'

c  Fixed quality version
      If (Version.EQ.0) THEN
         SumLmod=0.0
         do i=1,(Nmod-1)
            Lmod(i)=X(i)
            SumLmod=SumLmod+Lmod(i)
         end do
         Lmod(Nmod)=Lsegment-SumLmod
         hRo(Nmod)=X(Nmod)
         do i=1,Nmod
```

```
            pRo(i)=X(Nmod+i)
         end do

c  Fixed length version
      ELSE If (Version.EQ.1) THEN
         do i=1,Nmod
            hRo(i)=X(i)
            pRo(i)=X(Nmod+i)
         end do
      END IF

      TairCenter=X(Nvar)
      IF (Version.EQ.0) THEN
      CALL RegionFph(pRo(Nmod),hRo(Nmod),hgo(Nmod),hfo(Nmod),
     &      regOUT(Nmod))
      CALL XrFph(hRo(Nmod),pRo(Nmod),xRo(Nmod))
      END IF

c  Call subroutine HX

      k=SegNumb
      Qsegment(k)=0.0
      tAosum=0.0
      tAoAVGSegment=0.0
      IF (IQair.EQ.3) THEN
         do k=1,Nmod/2
            tAi(k)=TairCenter
            CALL HaFt(hAi(k),tAi(k))
         end do
      END IF

c  Evaluate air inlet temp to front and back sections of U-tube

      do i=1,Nmod
         CALL HX(i)
      end do
      IF (IQair.EQ.3) THEN
         do k=(Nmod/2+1),Nmod
            tAosum=tAosum+tAo(k)*(Lmod(k)/Lsegment*2)
         end do
      ELSE
         do k=1,Nmod
            tAosum=tAosum+tAo(k)*(Lmod(k)/Lsegment)
         end do
      END IF
      tAoAVGSegment=tAosum
      FF(Nvar)=TairCenter-(tAiSegment+TairFactor*(tAoAVGSegment
     *         -tAiSegment))

c  Transfer residuals from RES(,) to F():

      k=1
      do j=1,2
         do i=1,Nmod
            FF(k)=Res(j,i)
            k=k+1
         end do
      end do
```

```
      RETURN
      END

C ********  SOURCE: NWTRPH.FOR
************************************
C
C*************************************************************
C                                                             *
C           WRITTEN BY SUE MURLEY                             *
C           OPERATIONS RESEARCH LABORATORY                    *
C           DEPT. OF MECHANICAL AND INDUSTRIAL ENGINEERING    *
C           UNIVERSITY OF ILLINOIS AT URBANA/CHAMPAIGN        *
C*************************************************************
C
C
C ********  DESCRIPTION:
C         SUBROUTINES TO PERFORM THE NEWTON-RAPHSON
C         SOLUTION PROCEDURE
C
C
C
C ********  SUBROUTINES:
C         LINSYS   -  GAUSS-JORDAN LINEAR EQUATION SOLUTION
C                     AND MATRIX INVERSION
C         NWTRPH   -  PERFORMS NEWTON-RAPHSON ALGORITHM
C                     TO SOLVE A SYSTEM OF NONLINEAR
C                     EQUATIONS
C         FPRIME   -  EVALUATES THE PARTIAL DERIVATIVE
C                     MATRIX
C         MXMLT    -  MATRIX MULTIPLICATION ROUTINE
C         MXVMLT   -  MATRIX - VECTOR MULTIPLICATIONROUTINE
C         MXWRT    -  MATRIX PRINT ROUTINE
C         VCWRT    -  VECTOR PRINT ROUTINE
C         NAMWRT   -  PRINT NAME OF MATRIX/VECTOR
C
C
*****************************************************************
C
*****************************************************************
C
      SUBROUTINE LINSYS2(NDIM,N,A,B,ABT,AVCTR,ANV,XX,DET)
C ***
C *** GAUSS JORDAN LINEAR EQUATION SOLUTION AND MATRIXINVERSION
C *** (USING SCALED COLUMN PIVOTING)
C *** NDIM=ARRAY DIMENSION, N=MATRIX ORDER, A=N X N MATRIX,
C *** B=GIVEN VECTOR, AVCTR=ABT=PARAMETER(TYPICALLY 1.0E-06),
C *** ANV=INVERSE OF A, X=SOLUTION VECTOR,
C *** ABS(DET)=ABS(DETERMINANT OF SCALED A),-1<=DET=>1
C ***
      DIMENSION A(40,40),ANV(40,40),B(40),XX(40)
      DIMENSION D(50,50),LR(50),LC(50),SC(50)
C
C *** INITIALIZE ARRAYS TO ZERO
C
      KSING=0
      DO 10 L=1,N
      LR(L)=0
      LC(L)=0
```

```
      XX(L)=0.
   10 CONTINUE
C
      DO 18 J=1,N
         DO 15 K=1,N
            ANV(J,K)=0.
            D(J,K)=0.
   15    CONTINUE
         D(J,J)=1.
   18 CONTINUE
C
C *** COMPUTE COLUMN NORMS FOR SCALING
C
      NN=N
      DET=1.0
      DO 180 KC=1,N
         S=0.
         DO 140 KR=1,N
         S=S+A(KR,KC)**2
  140    CONTINUE
         SC(KC)=SQRT(S)
         IF(SQRT(S).LT.ABT) WRITE(6,1141) KC,SC(KC)
 1141    FORMAT(' LINSYS: COLUMN NORM .LT. TOLERANCE.COL,NORM: ',
     1          I6,E12.4)
C
C *** SCALE MATRIX ELEMENTS
C
c         DO 160 KR=1,N
c       A(KR,KC)=A(KR,KC)/SC(KC)
c  160    CONTINUE
  180 CONTINUE
C
C
C ********  START OF 200 LOOP  ************
C *** LOCATE POSITION OF PIVOT ELEMENT FOR ROW KK:
C *** MAXIMUM ELEMENT IN ROW
C
C *** PERFORM ERROR PROCESSING FOR THE FOLLOWING:
C ***    (1)  ZERO PIVOT ELEMENTS
C ***    (2)  PIVOTS SMALLER THAN SELECTED TOLERANCE, ABT
C ***          (MATRIX SINGULAR)
C
      DO 200 KK=1,N
       KKD=KK-1
       AMX=0.
       JPV=0
       KPV=0
       DO 20 J=1,N
       DO 22 L=1,N
          IF(J.EQ.LR(L)) GOTO 20
   22     CONTINUE
          DO 30 K=1,N
             IF(ABS(A(J,K)).LE.AMX) GOTO 30
             AMX=ABS(A(J,K))
             JPV=J
             KPV=K
   30  CONTINUE
   20  CONTINUE
```

```
C
C *** PRINT ERROR MESSAGE AND RETURN IF NO NON-ZERO PIVOTS
C
       IF(JPV.EQ.0) THEN
          WRITE (*,*)  ' LINSYS: ALL ELIGIBLE PIVOTS ARE ZERO'
          RETURN
       ENDIF
C
C *** CHECK SIZE OF PIVOT ELEMENTS
C
       TMI=A(JPV,KPV)
C
       IF(ABS(TMI).LT.ABT) THEN
       KSING=1
       WRITE(*,1130) TMI,ABT
 1130  FORMAT(' LINSYS: PIVOT SMALLER THAN ABT.PIVOT,ABT',2E12.4)
C
C
C *** PRINT SCALED MATRIX FOR DEBUG
C
       WRITE(*,1135) KKD,N,KK
 1135  FORMAT(' MATRIX SINGULAR,RANK=',I6,5X,'ORDER=',I6,5X,'KK=',I3)

       WRITE(*,1140) (L,LR(L),LC(L),B(L),L=1,KKD)
 1140  FORMAT(' PIVOT NUMBER, ROW, COLUMN, B',/3I6,E12.4)
       WRITE(*,*) ' PROCESSED SCALED MATRIX:'
C
       DO 112 JW=1,N
        WRITE(*,1145) JW,(A(JW,KW),KW=1,N)
 1145   FORMAT(1X,I5,10E12.4,(5X,10E12.4))
  112   CONTINUE
C
       DO 152 L=1,N
        DO 150 K=1,KKD
         IF (L.EQ.LR(K)) GOTO 150
  150 CONTINUE
          IF (ABS(B(L)).GE.AVCTR) THEN
             WRITE (*,*) ' MATRIX SINGULAR, EQUATIONS INCOMPATIBLE'
             GO TO 1000
          ENDIF
  152  CONTINUE
C
       WRITE (*,*) ' MATRIX SINGULAR AND EQUATIONS COMPATIBLE'
       WRITE (*,*) ' VECTOR X IS ONE SOLUTION'
       NN=KKD
       GOTO 202
C
     ENDIF
C
C *** END OF ERROR PROCESSING
C
C
C *** PERFORM GAUSS-JORDAN ALGORITHM
C
       DET=DET*TMI
       DO 40 K=1,N
          A(JPV,K)=A(JPV,K)/TMI
          D(JPV,K)=D(JPV,K)/TMI
```

```
   40  CONTINUE
       B(JPV)=B(JPV)/TMI
       A(JPV,KPV)=1.
       DO 70 I=1,N
          IF (I.NE.JPV) THEN
           TM2=A(I,KPV)
C
C *** SKIP IF ELEMENT ABOVE PIVOT = 0
C
           IF (TM2.NE.0.0) THEN
              DO 60 K=1,N
                 A(I,K)=A(I,K)-TM2*A(JPV,K)
                 D(I,K)=D(I,K)-TM2*D(JPV,K)
   60       CONTINUE
               B(I)=B(I)-TM2*B(JPV)
               A(I,KPV)=0.
            ENDIF
         ENDIF
   70  CONTINUE
       LR(KK)=JPV
       LC(KK)=KPV
  200 CONTINUE
C
C ********  END OF 200 LOOP  *************
C
C
  202 CONTINUE
C
C *** UNSCRAMBLE B INTO C AND D INTO ANV
C
      DO 90 L=1,NN
         L1=LR(L)
         L2=LC(L)
         XX(L2)=B(L1)
         IF (KSING.NE.1)THEN
            DO 80 LL=1,N
             ANV(L2,LL)=D(L1,LL)
   80       CONTINUE
         ENDIF
   90 CONTINUE
C
C *** UNSCALE X AND ANV
C
C      DO 400 KC=1,N
C       XX(KC)=XX(KC)/SC(KC)
C       DO 400 KR=1,N
C         ANV(KC,KR)=ANV(KC,KR)/SC(KC)
C 400   CONTINUE
C
C
      LRS=0
      LCS=0
      DO 120 L=1,NN
         LRS=LRS+LR(L)
         LCS=LCS+LC(L)
  120 CONTINUE
C
C
```

```
      IF(NN.NE.N) THEN
         LD=(NN*(NN+1))/2
         IF(LRS.NE.LD) WRITE(*,2005) LRS,LD
 2005    FORMAT(' GJLSC3 ROW SUM ERROR, LRS,LD',2I6)
         IF(LCS.NE.LD) WRITE(*,2010) LCS,LD
 2010    FORMAT(' GJLSC3 COLUMN SUM ERROR, LCS,LD',2I6)
      ENDIF
C
 1000 CONTINUE
      RETURN
      END
C
C
*****************************************************************
*****************************************************************
C
C
      SUBROUTINE MXMLT(NDIM,N1,N2,N3,A,B,AB)
      DIMENSION A(NDIM,NDIM),B(NDIM,NDIM),AB(NDIM,NDIM)
C ***
C *** MATRIX MULTIPLICATION ROUTINE
C ***
      DO 120 J=1,N1
      DO 120 K=1,N3
         S=0.
         DO 100 L=1,N2
         S=S+A(J,L)*B(L,K)
  100    CONTINUE
         AB(J,K)=S
  120 CONTINUE
      END
C
C
*****************************************************************
*****************************************************************
C
C
      SUBROUTINE MXVMLT(NDIM,N1,N2,A,V,AV)
C ***
C *** MATRIX X VECTOR MULTIPLICATION ROUTINE
C ***
      DIMENSION A(NDIM,NDIM),V(NDIM),AV(NDIM)
C
      DO 120 J=1,N1
      S=0.
      DO 100 K=1,N2
        S=S+A(J,K)*V(K)
  100 CONTINUE
      AV(J)=S
  120 CONTINUE
      END
C
C
*****************************************************************
*****************************************************************
C
C
      SUBROUTINE MXWRT(NDIM,N1,N2,A,NAME)
```

```
C ***
C *** MATRIX PRINT ROUTINE
C ***
      DIMENSION A(40,40)
      CHARACTER NAME*5
C
      WRITE(6,5)NAME
    5 FORMAT(' MATRIX ',A5)
C
      DO 100 J=1,N1
      WRITE(6,3)J,(A(J,K),K=1,N2)
    3 FORMAT(I4,/(5F17.9))
  100 CONTINUE
      RETURN
      END
C
C
*****************************************************************
*****************************************************************
C
C
      SUBROUTINE VCWRT(NDIM,N1,V,NAME)
C ***
C *** VECTOR PRINT ROUTINE
C ***
      DIMENSION V(NDIM)
      CHARACTER NAME*20
C
C     WRITE(6,'(A)') NAME
C     WRITE(6,3) (V(J),J=1,N1)
    3 FORMAT((5G12.5))
      WRITE (6,*) ' '
      RETURN
      END
C
C
*****************************************************************
*****************************************************************
C
C
      SUBROUTINE NAMWRT(NAME)
      CHARACTER NAME*5
C     WRITE(6,3)NAME
    3 FORMAT(' ARRAY',/A5)
      END
C
```

A.2 NIST Interface Source Code

```
      SUBROUTINE Initial
C
C  This subroutine is will initialize the constant data arrays
C  to run the NIST subroutines.  This is for the NIST CSD
C  Version 3.0 routines.
C
C  UTS - Set units for program
C      1 - SI units    K, kg/m^3, KPa, kJ/kg, kJ/kg K
C      2 - Eng units   F, lbm/ft^3, psi, Btu/lbm, Btu/lbm F
C
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS,MFLAG
      CHARACTER HREF(0:40)*6, HNAME(40)*48, REGNO*9
      DIMENSION FIN(5,5),X(5)
      INCLUDE 'NISTCOM'

      NCIN=1
      UTS=2
      IR(1)=16
      XV(1)=1.0
      XL(1)=1.0

      CALL BCONST(NCIN,IR,FIN)

      RETURN
      END

      SUBROUTINE PsatT(Temp,satP)
C
C  Subroutine which calculates the saturation pressure given
C  the temperature.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT

      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         T=(Temp+459.67)/1.8
      ELSE
         T=Temp
      END IF

      CALL BUBLT(T,XL,XV,P,VL,VV,.TRUE.,LCRIT)

      IF (UTS.EQ.2) THEN
         satP=P/6.894757
      ELSE
         satP=P
      END IF

      RETURN
      END
```

```
      SUBROUTINE TsatP(Press,satT)
C
C  Subroutine which calculates the Saturation temperature
C  given the pressure.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
      ELSE
         P=Press
      END IF
      CALL BUBLP(P,XL,XV,T,VL,VV,.TRUE.,LCRIT)
      IF (UTS.EQ.2) THEN
         satT=(T*1.8)-459.67
      ELSE
         satT=T
      END IF

      RETURN
      END

      SUBROUTINE HPT(Press,Temp,Enth)
C
C  This subroutine calculates the enthalpy given the pressure
C  and temperature in either the subcooled or superheated
C  regions.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
         T=(Temp+459.67)/1.8
      ELSE
         P=Press
         T=Temp
      END IF
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      IF (T.GT.Tsat) THEN
         CALL BUBLP(P,XL,XV,Tsat,VL,VV,.FALSE.,LCRIT)
         CALL ESPAR(0,T,XV,A1,B1)
         CALL VIT(T,P,A1,B1,VV,.FALSE.,LVCON)
         IF (LVCON) THEN
          WRITE(*,*) 'VIT not converging'
       END IF
       CALL HCVCPS(1,T,VV,XV,H,CV,CP,VS)
      ELSE IF (T.LT.Tsat) THEN
         CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
         CALL ESPAR(0,T,XL,A1,B1)
         CALL VIT(T,P,A1,B1,VL,.TRUE.,LVCON)
         IF (LVCON) THEN
```

```
            WRITE(*,*) 'VIT not converging'
         END IF
         CALL HCVCPS(1,T,VL,XL,H,CV,CP,VS)
      END IF

      IF (UTS.EQ.2) THEN
         Enth=(H/2.33)/CRIT(1,IR(1))
      ELSE
         Enth=(H/CRIT(1,IR(1)))
      END IF

      RETURN
      END

      SUBROUTINE HPX(Press,Qual,Enth)
C
C  Subroutine to determine the enthalpy given the pressure
C  and quality.  Only valid for the Saturation region.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
      ELSE
         P=Press
      END IF
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.FALSE.,LCRIT)
      CALL HCVCPS(1,Tsat,VV,XV,HV,CV,CP,VS)
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      CALL HCVCPS(1,Tsat,VL,XL,HL,CV,CP,VS)
      H=HL+(HV-HL)*Qual

      IF (UTS.EQ.2) THEN
         Enth=(H/2.33)/CRIT(1,IR(1))
      ELSE
         Enth=(H/CRIT(1,IR(1)))
      END IF

      RETURN
      END

      SUBROUTINE VPT(Press,Temp,Vol)
C
C  This subroutine determines the specific volume given the
C  pressure and temperature
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
         T=(Temp+459.67)/1.8
      ELSE
```

```
            P=Press
            T=Temp
         END IF
         CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
         IF (T.GT.Tsat) THEN
            CALL BUBLP(P,XL,XV,Tsat,VL,VV,.FALSE.,LCRIT)
            CALL ESPAR(0,T,XV,A1,B1)
            CALL VIT(T,P,A1,B1,VV,.FALSE.,LVCON)
            IF (LVCON) THEN
             WRITE(*,*) 'VIT not converging'
          END IF
          V=VV
         ELSE IF (T.LT.Tsat) THEN
            CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
            CALL ESPAR(0,T,XL,A1,B1)
            CALL VIT(T,P,A1,B1,VL,.TRUE.,LVCON)
            IF (LVCON) THEN
               WRITE(*,*) 'VIT not converging'
            END IF
            V=VL
         END IF
         IF (UTS.EQ.2) THEN
            Vol=(V/CRIT(1,IR(1)))*16.018463
         ELSE
            Vol=V/CRIT(1,IR(1))
         END IF

         RETURN
         END

        SUBROUTINE Saturation(Press,Tsat,Vg,Vf,Hg,Hf,Hfg,Sg,Sf)
C
C  This subroutine calculates the saturation properties
C  given the pressure.
C
        IMPLICIT DOUBLE PRECISION(A-H,O-Z)
        INTEGER UTS
        LOGICAL LCRIT,LVCON
        DIMENSION X(5)
        INCLUDE 'NISTCOM'

        IF (UTS.EQ.2) THEN
           P=Press*6.894757
        ELSE
           P=Press
        END IF
        X(1)=1.0D0
        CALL BUBLP(P,XL,XV,T,VL,VV,.FALSE.,LCRIT)
        CALL HCVCPS(1,T,VV,XV,HV,CV,CP,VS)
        CALL HCVCPS(1,T,VL,XL,HL,CV,CP,VS)
        Sg=ENTROP(T,VV,X)
        Sf=ENTROP(T,VL,X)

        IF (UTS.EQ.2) THEN
           Vg=(VV/CRIT(1,IR(1)))*16.018463
           Vf=(VL/CRIT(1,IR(1)))*16.018463
           Hg=(HV/2.33)/CRIT(1,IR(1))
           Hf=(HL/2.33)/CRIT(1,IR(1))
```

```
            Hfg=Hg-Hf
            Tsat=(T*1.8)-459.67
            Sg=Sg*0.23901/CRIT(1,IR(1))
            Sf=Sf*0.23901/CRIT(1,IR(1))
         ELSE
            Vg=VV/CRIT(1,IR(1))
            Vf=VL/CRIT(1,IR(1))
            Hg=HV/CRIT(1,IR(1))
            Hf=HL/CRIT(1,IR(1))
            Hfg=Hg-Hf
            Sg=Sg/CRIT(1,IR(1))
            Sf=Sf/CRIT(1,IR(1))
         END IF

         RETURN
         END


         SUBROUTINE TPH(Press,Enth,Temp)
C
C  This subroutine calculates the temperature given the pressure
C  and enthalpy in any region.
C
         IMPLICIT DOUBLE PRECISION(A-H,O-Z)
         INTEGER UTS
         DIMENSION X(5),XLL(5),XVV(5)
         INCLUDE 'NISTCOM'

         IF (UTS.EQ.2) THEN
            P=Press*6.894757
            H=Enth*2.33*CRIT(1,IR(1))
         ELSE
            P=Press
            H=Enth*CRIT(1,IR(1))
         END IF
         X(1)=1.0
         CALL HPIN(H,P,X,T,XQ,XLL,XVV,VL,VV,HL,HV)
         IF (XQ.LT.0.0) THEN
            CALL HPXSP(H,P,X,T,HL,VL,T2,V,.TRUE.)
            T=T2
         ELSE IF (XQ.GT.1.0) THEN
            CALL HPXSP(H,P,X,T,HV,VV,T2,V,.FALSE.)
            T=T2
         END IF
         IF (UTS.EQ.2) THEN
            Temp=(T*1.8)-459.67
         ELSE
            Temp=T
         END IF

         RETURN
         END

         SUBROUTINE XPH(Press,Enth,Qual)
C
C  This subroutine calculates the quality given the pressure and
C  enthalpy.
C
```

```
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      DIMENSION X(5),XLL(5),XVV(5)
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
         H=Enth*2.33*CRIT(1,IR(1))
      ELSE
         P=Press
         H=Enth*CRIT(1,IR(1))
      END IF
      X(1)=1
      CALL HPIN(H,P,X,T,XQ,XLL,XVV,VL,VV,HL,HV)
      Qual=XQ

      RETURN
      END

      SUBROUTINE CvPT(Press,Temp,Cv,Cp,Vs)
C
C  Subroutine to find the constant volume and pressure specific heat
C  along with the velocity of sound given the temperature and pressure.
C  Valid only for the superheated vapor and subcooled liquid regions.
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
         T=(Temp+459.67)/1.8
      ELSE
         P=Press
         T=Temp+273.15
      END IF
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      IF (T.GT.Tsat) THEN
         CALL BUBLP(P,XL,XV,Tsat,VL,VV,.FALSE.,LCRIT)
         CALL ESPAR(0,T,XV,A1,B1)
         CALL VIT(T,P,A1,B1,VV,.FALSE.,LVCON)
         IF (LVCON) THEN
            WRITE(*,*) 'VIT not converging'
         END IF
         CALL HCVCPS(5,T,VV,XV,H,CV,CP,VS)
      ELSE IF (T.LT.Tsat) THEN
         CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
         CALL ESPAR(0,T,XL,A1,B1)
         CALL VIT(T,P,A1,B1,VL,.TRUE.,LVCON)
         IF (LVCON) THEN
            WRITE(*,*) 'VIT not converging'
         END IF
         CALL HCVCPS(5,T,VL,XL,H,CV,CP,VS)
      END IF

      IF (UTS.EQ.2) THEN
         Cv=CV*0.23901/CRIT(1,IR(1))
```

```
         Cp=CP*0.23901/CRIT(1,IR(1))
         Vs=VS*3.2808
      ELSE
         Cv=CV/CRIT(1,IR(1))
         Cp=CP/CRIT(1,IR(1))
      END IF
      RETURN
      END


      SUBROUTINE CvPsat(Press,CvL,CpL,CvV,CpV,VsV)
c
c  Subroutine to find the constant volume and pressure specific heat
c  along with the velocity of sound for the saturated vapor and
c  liquid give the saturation pressure.
c
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
      ELSE
         P=Press
      END IF
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.FALSE.,LCRIT)
      CALL HCVCPS(5,Tsat,VV,XV,HV,CVV,CPV,VSV)
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      CALL HCVCPS(5,Tsat,VL,XL,HL,CVL,CPL,VS)

      IF (UTS.EQ.2) THEN
         CvL=CVL*0.23901/CRIT(1,IR(1))
         CpL=CPL*0.23901/CRIT(1,IR(1))
         CvV=CVV*0.23901/CRIT(1,IR(1))
         CpV=CPV*0.23901/CRIT(1,IR(1))
         VsV=VSV*3.2808
      ELSE
         CvL=CVL/CRIT(1,IR(1))
         CpL=CPL/CRIT(1,IR(1))
         CvV=CVV/CRIT(1,IR(1))
         CpV=CPV/CRIT(1,IR(1))
      END IF
      RETURN
      END


      SUBROUTINE CvTsat(Temp,CvL,CpL,CvV,CpV,VsV)
c
c  Subroutine to find the constant volume and pressure specific heat
c  along with the velocity of sound for the saturated vapor and
c  liquid given the saturation temperature.
c
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      INCLUDE 'NISTCOM'
```

```
      IF (UTS.EQ.2) THEN
         T=(Temp+459.67)/1.8
      ELSE
         T=Temp
      END IF
      CALL BUBLT(T,XL,XV,Psat,VL,VV,.FALSE.,LCRIT)
      CALL HCVCPS(5,T,VV,XV,HV,CVV,CPV,VSV)
      CALL BUBLT(T,XL,XV,Psat,VL,VV,.TRUE.,LCRIT)
      CALL HCVCPS(5,T,VL,XL,HL,CVL,CPL,VS)

      IF (UTS.EQ.2) THEN
         CvL=CVL*0.23901/CRIT(1,IR(1))
         CpL=CPL*0.23901/CRIT(1,IR(1))
         CvV=CVV*0.23901/CRIT(1,IR(1))
         CpV=CPV*0.23901/CRIT(1,IR(1))
         VsV=VSV*3.2808
      ELSE
         CvL=CVL/CRIT(1,IR(1))
         CpL=CPL/CRIT(1,IR(1))
         CvV=CVV/CRIT(1,IR(1))
         CpV=CPV/CRIT(1,IR(1))
      END IF
      RETURN
      END


      SUBROUTINE SPT(Press,Temp,S)
c
c  Subroutine to find the entropy given the temperature and pressure
c  valid for subcooled liquid or superheated vapor.
c
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      DIMENSION X(5)
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
         T=(Temp+459.67)/1.8
      ELSE
         P=Press
         T=Temp
      END IF
      X(1)=1.0
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      IF (T.GT.Tsat) THEN
         CALL ESPAR(0,T,XV,A1,B1)
         CALL VIT(T,P,A1,B1,VV,.FALSE.,LVCON)
         IF (LVCON) THEN
            write(*,*) 'VIT not converging'
         END IF
         S=ENTROP(T,VV,X)
      ELSE IF (T.LT.Tsat) THEN
         CALL ESPAR(0,T,XL,A1,B1)
         CALL VIT(T,P,A1,B1,VL,.TRUE.,LVCON)
         IF (LVCON) THEN
            write(*,*) 'VIT not converging'
```

```
         END IF
         S=ENTROP(T,VL,X)
      END IF
      IF (UTS.EQ.2) THEN
         S=S*0.23901/CRIT(1,IR(1))
      ELSE
         S=S/CRIT(1,IR(1))
      END IF

      RETURN
      END

      SUBROUTINE SPX(Press,Qual,S)
c
c  Subroutine to calculate the entropy given the pressure
c  and quality.  Valid only for the saturation region.
c
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      INTEGER UTS
      LOGICAL LCRIT,LVCON
      DIMENSION X(5)
      INCLUDE 'NISTCOM'

      IF (UTS.EQ.2) THEN
         P=Press*6.894757
      ELSE
         P=Press
      END IF
      X(1)=1.0
      CALL BUBLP(P,XL,XV,Tsat,VL,VV,.TRUE.,LCRIT)
      Sg=ENTROP(Tsat,VV,X)
      Sf=ENTROP(Tsat,VL,X)
      S=Sf+(Sg-Sf)*Qual
      IF (UTS.EQ.2) THEN
         S=S*0.23901/CRIT(1,IR(1))
      ELSE
         S=S/CRIT(1,IR(1))
      END IF

      RETURN
      END
```

APPENDIX B - EXPERIMENTAL DATA

B.1 Complete Data Set

This appendix provides a table of the operating conditions for the 46 experimental data points used in the results comparison.

No.	Ref. Inlet Temp. °F	Ref. Inlet Pressure psi	Air Inlet Temp. °F	Air Inlet Pressure psi	Relative Humid.	Ref. Flow Rate lbm/hr	Air Flow Rate lbm/hr
1	137.60	200.36	79.90	14.48	0.39	197.43	1838.17
2	124.00	165.96	80.00	14.48	0.47	194.64	3929.41
3	123.20	155.36	79.90	14.48	0.53	192.79	6178.17
4	181.60	304.76	79.80	14.48	0.40	384.97	1848.88
5	151.70	230.96	80.20	14.48	0.47	384.78	3920.68
6	145.50	206.56	79.80	14.48	0.54	389.31	6509.75
7	165.40	274.56	78.80	14.48	0.35	560.34	4947.51
8	154.80	249.66	80.00	14.48	0.35	568.15	7524.73
9	218.50	205.16	80.10	14.48	0.41	156.23	1698.57
10	224.40	167.76	80.10	14.48	0.47	153.11	4092.59
11	223.10	154.06	80.00	14.48	0.53	150.13	7089.33
12	244.20	344.96	80.70	14.48	0.39	381.13	1891.60
13	219.50	254.26	77.60	14.34	0.45	400.95	4295.77
14	212.20	226.56	79.10	14.34	0.43	396.36	6850.33
15	230.00	468.76	79.80	14.48	0.38	668.50	1757.42
16	197.10	289.26	76.40	14.48	0.38	555.13	4740.54
17	197.40	268.66	79.90	14.48	0.36	568.09	7511.98
18	154.10	249.96	110.20	14.48	0.30	154.92	1984.28
19	138.20	220.36	110.00	14.48	0.30	154.84	4041.47
20	154.60	219.56	110.00	14.48	0.26	184.64	6754.79
21	168.30	325.26	110.00	14.51	0.27	402.22	2972.23
22	162.40	276.76	100.10	14.48	0.33	416.11	4795.47
23	160.70	272.86	110.00	14.53	0.30	387.35	6999.58
24	182.60	346.86	110.10	14.48	0.26	548.60	4915.26
25	171.60	316.96	109.90	14.48	0.26	563.13	7627.65

26	243.50	240.06	110.00	14.48	0.30	149.80	4043.73
27	223.50	231.56	110.00	14.48	0.26	182.61	6742.96
28	213.30	340.86	97.20	14.39	0.43	386.57	2315.96
29	220.40	297.66	100.00	14.48	0.33	400.69	4791.98
30	212.80	274.36	100.00	14.48	0.33	395.07	6585.46
31	232.90	378.26	110.00	14.48	0.26	539.71	4905.90
32	216.60	340.36	110.00	14.48	0.26	556.57	7658.51
33	188.20	379.36	150.00	14.48	0.22	154.50	2003.87
34	183.40	345.26	150.00	14.48	0.22	155.77	4953.24
35	182.10	334.76	150.30	14.48	0.21	156.76	8020.68
36	206.80	465.16	150.10	14.48	0.22	366.59	2623.79
37	190.00	429.16	150.10	14.48	0.22	391.95	3939.23
38	197.00	370.46	140.10	14.53	0.23	380.27	6955.05
39	188.10	398.56	139.90	14.53	0.23	517.62	7008.56
40	263.40	405.26	150.00	14.48	0.22	150.87	2197.14
41	258.50	365.76	150.00	14.48	0.22	153.96	4398.08
42	230.30	308.66	140.10	14.48	0.22	153.19	8187.77
43	254.40	451.06	149.10	14.53	0.21	370.91	4738.97
44	262.10	399.26	140.20	14.53	0.23	379.41	6935.49
45	246.90	483.56	150.10	14.39	0.22	484.70	4918.50
46	244.50	428.16	139.90	14.53	0.23	494.98	6943.45

B.2 Search Data Set

This appendix provides the operating conditions for the twenty data points used in the parameter estimation searches.

No.	Ref. Inlet Temp. °F	Ref. Inlet Pressure psi	Air Inlet Temp. °F	Air Inlet Pressure psi	Relative Humid.	Ref. Flow Rate lbm/hr	Air Flow Rate lbm/hr
1	137.60	200.36	79.90	14.48	0.39	197.43	1838.17
2	124.00	165.96	80.00	14.48	0.47	194.64	3929.41
4	181.60	304.76	79.80	14.48	0.40	384.97	1848.88
5	151.70	230.96	80.20	14.48	0.47	384.78	3920.68
13	219.50	254.26	77.60	14.34	0.45	400.95	4295.77
14	212.20	226.56	79.10	14.34	0.43	396.36	6850.33
18	154.10	249.96	110.20	14.48	0.30	154.92	1984.28
19	138.20	220.36	110.00	14.48	0.30	154.84	4041.47
20	154.60	219.56	110.00	14.48	0.26	184.64	6754.79
21	168.30	325.26	110.00	14.51	0.27	402.22	2972.23
23	160.70	272.86	110.00	14.53	0.30	387.35	6999.58
24	182.60	346.86	110.10	14.48	0.26	548.60	4915.26
25	171.60	316.96	109.90	14.48	0.26	563.13	7627.65
28	213.30	340.86	97.20	14.39	0.43	386.57	2315.96
30	212.80	274.36	100.00	14.48	0.33	395.07	6585.46
36	206.80	465.16	150.10	14.48	0.22	366.59	2623.79
37	190.00	429.16	150.10	14.48	0.22	391.95	3939.23
38	197.00	370.46	140.10	14.53	0.23	380.27	6955.05
45	246.90	483.56	150.10	14.39	0.22	484.70	4918.50
46	244.50	428.16	139.90	14.53	0.23	494.98	6943.45

APPENDIX C - SIMULATION INSTRUCTIONS

This appendix outlines the instructions for running the simulation program with the exhaustive search routine. The input files are detailed and examples are provided.

A. Input Files - The input files for the Module Based Condenser Simulation Program allow for the set up of coils with different geometries. The input files used by the program are:

1. TestData - File containing the experimental data. The required information for this file is:

   Line 1 - Number of data points in file.

   Line 2 - Number of entries per data point (10 for the exhaustive search)

   Line 3 - Experimental Data Values in the following order:

   Ticr, Picr, Tica, Pica, RH, Mref, Mair, Qexp, Pocr, dTsub

   Ticr - Condenser Refrigerant Inlet Temperature
   Picr - Condenser Refrigerant Inlet Pressure
   Tica - Condenser Air Inlet Temperature
   Pica - Condenser Air Inlet Pressure
   RH - Condenser Air Inlet Relative Humidity
   Mref - Condenser Refrigerant Mass Flow Rate
   Mair - Condenser Air Mass Flow Rate
   Qexp - Condenser Experimental Heat Transfer
   Pocr - Condenser Refrigerant Outlet Pressure
   dTsub - Condenser Degrees of Subcooling at Exit

   Line # - Repeat Line 3 for all additional data points.

2. QorLSegment - File containing the segment information for fixed quality or length.

Line 1 - Number of segments in coil.

Line 2 - Number of modules in segment one.

Line 3 - Exit quality or length of each module in the segment.

Line # - Repeat Lines 2 and 3 for any additional segments.

3. SegmentInfo - File containing information regarding each segment.

Line 1 - Number of segments in coil.

Line 2 - Number of input values for each segment (8).

Line 3 - Enter information for each segment:

LSegment, mRSegment, Version, Tairfactor, Nmod, IQair, IQref, IQpd

LSegment - Total length of segment (ft).
mRSegment - Refrigerant mass flow rate of segment as ratio of total flow rate.
Version - Fixed length or quality version.
Tairfactor - Air outlet temperature adjustment.
Nmod - Number of modules in segment.
IQair - Input air qualifier
- 1 - Air temperature into segment equals temperature into condenser
- 2 - Air temperature into segment equals average outlet temperature of previous segment
- 3 - Air temperature into segment is temperature into condenser for front half of modules and average outlet temperature for back half of modules.

IQref - Input refrigerant qualifier
- 1 - Refrigerant properties into segment are inlet condenser refrigerant properties
- 2 - Refrigerant properties into segment are the previous segments outlet conditions

IQpd - Pressure drop input qualifier
- 0 - No manifold or return bend at segment

inlet
1 - Manifold at segment inlet
2 - Return bend at segment inlet

4. TestInfo - File containing general information regarding the condenser.

   Line 1 - Comment, Pchange, Refrigerant, Ltotal, Lfrontal, dzdl, ktube, Di, Do

   Comment - Selects which comments are printed.
   Pchange - Selects whether to include pressure drop.
   Refrigerant - Selects refrigerant (2 = R134a).
   Ltotal - Total condenser length.
   Lfrontal - Frontal condenser length.
   dzdl - Gravitational pressure drop per unit length.
   ktube - Tube thermal conductivity.
   Di - Tube inside diameter (ft).
   Do - Tube outside diameter (ft).

5. Parameters - File containing parameter initial guesses.

   Line 1 - Number of parameters.

   Line 2 - Parameter initial guesses.

6. Step - File containing initial step sizes for the parameters in the exhaustive search.

B. Select objective function in subroutine "objective1".

C. Compile and run program.